DOCUMENT RESUME

ED 024 602

SE 005 423

By-Kemeny, John G.; Kurtz, Thomas E.

The Dartmouth Time-Sharing Computing System. Final Report.

Spons Agency-National Science Foundation, Washington, D.C.

Pub Date Jun 67

Grant-NSF-GE-3864

Note-76p.

EDRS Price MF-$0.50 HC-$3.90

Descriptors-*Computer Assisted Instruction, *Computers, Curriculum, Engineering, *Higher Education, *Instruction, Mathematics, Physics, Program Descriptions, Psychology

Identifiers-Dartmouth College, National Science Foundation

Reported are the activities involved in introducing computer programing to students at Dartmouth College as part of their program in liberal education. How this project was accomplished and the subsequent impact on students in mathematics, engineering, psychology, physics, and business administration is presented in summary form. Also reported is the impact that this project has had on the faculty at Dartmouth and at other institutions. (RP)

EDO 24602

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

THE DARTMOUTH TIME-SHARING COMPUTING SYSTEM

developed under a grant from the

Course Content Improvement Program
National Science Foundation

(Grant NSF · GE-3864)

Final Report

June 1967

John G. Kemeny
Project Director

Thomas E. Kurtz
Assoc. Project Director

SE005 423

THE DARTMOUTH TIME-SHARING COMPUTING SYSTEM

developed under a grant from the

Course Content Improvement Program
National Science Foundation

(Grant NSF · GE-3864)

Final Report
April 1967

John G. Kemeny
Project Director

Thomas E. Kurtz
Assoc. Project Director

TABLE OF CONTENTS

		Page
1.	The Goal of the Project	1
2.	What was Accomplished	5
3.	Impact on the Student Body	10
	Mathematics	11
	Engineering	13
	Psychology	15
	Physics	17
	Business Administration	17
4.	Impact on the Faculty	20
5.	Impact on Other Institutions	23
	Use by Colleges	23
	Use by Secondary Schools	24
	An Educational Utility	26
	Commercial Use	27
	Future Plans	28

APPENDICES

I. The Time-Sharing System

II. The Language BASIC

III. The Freshman Training Program

TEACHING SUPPLEMENTS (not included)

I. Use of the Computer in a Course in Number Theory

II. Computer Programs of Use in Elementary Statistics

III. Computer Programs of Use in Statistics

IV. Use of the Computer in a Course in Logic

V. Using the Computer in the Teaching of Constructive Linear Algebra

1. The Goal of the Project.

Four years ago Dartmouth College reached the conclusion that learning to use a high-speed computer should be an essential part of liberal education. Four years ago this was merely a dream, and considered impractical by many experts. Today it is a reality.

Computers are beginning to have an increasing effect on the lives of all of us. They play key roles in business, industry, government, and all forms of research. The average college graduate of today is almost sure to need a computer in his work twenty years from now. Therefore, we must prepare him today to use this most powerful of tools.

Even more significant is the need for changing the attitude of the typical intelligent person towards computers. Present attitudes are often a mixture of fear and superstitious awe. The same person may refuse to believe that computers can do large routine tasks, which are no more than simple exercises on a high-speed computer, and yet accept uncritically "conclusions" obtained by means of a computer. It is vitally important that the leaders of government, industry and education should know both the potential and limitations of the use of computers, and to be aware of the respective roles of Man and machine in the partnership.

By the end of the current academic year, Dartmouth will have introduced some 2000 students to the use of computers. This represents 80% of three freshman classes. All of these students will have completed four significant exercises, in which they personally "debug" their own programs. A significant minority of these students have acquired the habit of regular use of the Dartmouth system. But even those students who did only the minimal required work have changed their attitude towards computers. Anyone who tries to convince a Dartmouth undergraduate either that computers are to be feared or that they are of little use, will be met with well-founded scorn. The Dartmouth student knows better -- and knows it from personal experience.

A secondary goal was to put the computer at the fingertips of the Faculty. Four years ago only a handful of faculty members had ever used a computer. This was due in part to the nature of research on our campus. None of the faculty members were engaged in projects which required hours of computer time. And while many could have made good use of smaller quanta of computing time (and do make such use today), they found the nuisance of using a traditional computation center too great a deterrent. And none of us saw how these facilities could reasonably support our teaching.

Today some 40% of the faculty use the Dartmouth system. Faculty use represents a wide variety of projects. In the area of research we find anything from the computation of examples through data-analysis to large computer based research projects. And in teaching it is common to see a faculty member running off an illustration for his lecture -- usually five minutes before class. But he will feel equally free to ask his students to work significant exercises and term projects on the computer. He may be sure that his students know how to use a computer, that the computer will be available, and that the student will enjoy the assignment.

There are two reasons for our success: (1) Our easy-to-use time-sharing system, and (2) the new simple language BASIC. To understand the full significance of these developments, we must look at the state of the art of computing before time-sharing.

Five years ago the method of operation at computation centers was "batch-processing", and this is still the only procedure at most centers. It is designed to maximize the use of the computer for a large number of significant problems -- which are known to be correctly programmed. Since computers are much faster than human beings, it is inefficient to let a single human being use the machine all on his own. It takes too long for him to tell the computer what he wants, and it takes too long for him to understand the answer. Therefore human beings submit their requests, which are collected until there is a large number of them, and then these are fed into the machine -- usually from a magnetic tape. The computer works on the first problem for as long as necessary, and writes the answers on another tape. It can then immediately start work on the second problem. When all the problems have received their share of the computer's attention, the tape is transcribed on a printer, and the answers are distributed to the users. The elapsed time may be anywhere from two hours to a whole day.

A typical user may have received only a minute of the computer's time. But since this may save him years of work, it is well worth waiting for several hours until he can obtain his results, if they are correct. But human beings are not infallible, and the chances are that there were some slight mistakes in the instructions (in the program). The user must then make a correction, and resubmit his problem. It is quite common to have to do this ten or more times until the program finally works. Thus it may take two weeks of rather frustrating work until the programmer succeeds. During all this time the use of the computer is optimized -- an error may waste only a fraction of a second of the machine's time -- but it certainly is far from ideal for the human being.

We came to the conclusion that while research scientists may put up with this kind of treatment, Dartmouth undergraduates would rebel. We therfore designed a system that is ideal from the human point of view -- and yet is a surprisingly efficient use of the computer as well.

It would be impossible to have 250 students, during a ten week term, queuing up for their many dozens of attempts at making programs work. They would have to arrange their entire schedule at the computer's eonvenience. And they would often have to wait overnight until a simple typing error is corrected. These same factors play a major role in faculty reluctance to learn the use of computers.

Yet we could not afford to turn a million dollar machine over to a single user. Hence the answer is that the computer should serve a large number of users all at the same time. On our present system about 35 users can get good simultaneous service, each with the illusion that he has complete control of the machine. Each user sits at a teletype typewriter, types out his program, and keeps entering corrections until his program finally works. This makes it both convenient and pleasant to use the computer. The nature of this time-sharing system is discussed in the next section, and more technical details are furnished in Appendix I.

The second major obstacle to the use of computers was the necessity of learning a strange new language. "Machine language" is strictly for experts, and useless for the novice or for the casual user. The first break-through came about a decade ago, with the development of FORTRAN. It was the first language designed to make it as easy for the average user to write a program. Once he writes a program in FORTRAN, the computer itself translates it from FORTRAN into machine language, so that the user is under the illusion that the computer speaks FORTRAN. While FORTRAN was a major advance, it still has many disadvantages for the novice and the occasional user, and we decided that we could improve it.

Thus we decided to write a "Beginners' All-purpose Symbolic Instruction Code" and BASIC was born. Our one major mistake was to use the word 'Beginners' in the name, since now the language is widely used (and preferred) by experts as well as novices.

BASIC was originally designed as an extremely simple language that can be quickly mastered by a novice. This core has been retained. Our freshmen learn to program in BASIC in two one-hour lectures. These two lectures, together with a very brief manual, enable them to be on their on the computer. After that they learn from their own experience.

The language has since been expanded into a powerful, general purpose programming language. However unlike other languages, BASIC provides power without handicapping the novice. The beginner is simply not aware of the full power of the language. Whenever new instructions required that someone had to do extra work, we always put the burden on the experienced programmer.

A language like FORTRAN or ALGOL requires that the user remember a large number of conventions and that he specify a wide variety of options. This makes it harder to learn the language, and occasional users complain that they have to "relearn" the language each time. In BASIC this difficulty has been avoided. The language is as close to ordinary English combined with elementary algebra as possible. Whenever conventions are necessary or options available, a simple choice is automatically provided for the novice. The expert may specify a more advance option, if he desires.

It is tremendously encouraging to the novice, be he a student or a member of the Faculty, that he can do something interesting on the computer during the first week of the training program. The reader who may be sceptical on this point is encouraged to look at Appendix II, which discusses BASIC and furnishes examples.

The combination of time-sharing and BASIC has made our computation center perhaps the most popular educational/research facility on the campus. Everything we had hoped for four years ago has come true, and in many ways our success has far surpassed our dreams. We will speak of this in Sections 3 and 4.

But our most pleasant surprise has been wide national, and even international, attention attracted by the Dartmouth system. We have been visited by representatives of some 200 institutions who wished to see for themselves what we had accomplished. Many of these institutions have since become active users of time-sharing or are planning to install time-sharing systems patterned at least in part on ours. We will describe some of these uses in Section 5.

2. What was Accomplished.

The project officially started in September 1963. The staff consisted of the Director and Associate Director, both working part time, and a dozen very able Dartmouth undergraduates. The computer equipment had been ordered from the General Electric Company, but it would not be available for another 6 months.

During these months the detailed planning took place. It was decided early that the time-sharing system could best be implemented by using two computers, one to do the actual computations and one to carry on simultaneous conversations with all the users. The former was to be a GE-235, a reasonably fast but small computer. It had a 6 micro-second cycle time, and floating point operations take 30-60 micro-seconds. That is fast, but not nearly as fast as today's best computers. Its memory consists of only 16,000 20-bit words, which is a very small memory for a modern computer.

One of the main reasons for choosing GE equipment was the availability of the Datanet-30 for communication purposes. This is an independent computer in its own right, strong on logic and weak on arithmetic. It does, however, have a 16,000 word memory of its own. Since the small memory of the 235 was needed for the actual computations, most of the logic of the time-sharing system -- the so-called "executive program" -- had to be in the Datanet-30. This meant that the Datanet was the boss, and the 235 acted as a slave computer, simply following orders.

The third component of the system is a large random-access disc memory, capable of storing 6 million 20-bit words. This is used as scratch-paper, to accumulate the programs of all the current users, to hold compilers and library programs, and to provide the capability of storing programs for future use.

While the usual peripherals (printer, tapes, card reader and punch) were available, they do not play an important role during time-sharing. The essence of the system is the interaction between the two computers, and their joint use of the disc memory.

The most difficult design problem was the communication between two independent computers for which we had no precedents. This problem was successfully solved by Michael Busch and John McGeechie, a Dartmouth Sophomore working with a Dartmouth Junior. They coded and debugged the entire executive system.

At the same time the new language BASIC had to be designed from scratch, and a compiler had to be written. A "compiler" is a program that translates from a user language (BASIC in this case) to the language of the machine. A good compiler will enable the user to talk to the machine in BASIC, to have various grammatical errors spotted automatically, and will yield an efficient machine code.

In addition a wide variety of "utility routines" had to be written. We will mention only one of these. We wished to make the process of debugging as painless as possible. Therefore, we hit on the following idea: It is terribly difficult to make corrections in a machine-language program. Instead, our system would always save the original program in BASIC, and all corrections would be in that language. All instructions were to be numbered, and if a line was to be changed, one simply retyped the same line number with the corrected line. If an instruction was to be inserted between lines 30 and 40, one simply typed a new line with an intermediate line number, say 44. Therefore, we needed a utility routine that would take a program with corrections, and make the insertions and replacements automatically.

The computer (originally a slower 225) arrived on March 1, 1964. During the month of March it was undergoing shake-down, and some admustments had to be made in the hardware, But the programmers used every available moment for debugging the various soft-ware systems. On April 1 the system was officially turned over to Dartmouth. At 4 a.m. on May 1, 1964 three teletypes time-shared programs in BASIC. We are rather proud of that time-table.

But the hardest debugging was still before us. BASIC was in a rudimentary form. The executive system still was prone to catastrophies. Many more features were needed to make the system both fast enough to handle a large number of users and convenient enough for our purposes. We also had to prepare for the change-over to the 235. This faster computer arrived in September 1964, and in October we launched the first full-scale freshman training program.

Before proceeding, it will be useful to describe how our system looks to a user. We will not at this time talk about technical details, these are contained in Appendix I.

Let us observe a student who has just sat down at a teletype. He types 'HELLO', to tell the machine that there is a new user. He is then asked to identify himself, and then he is ready to type in a program. Let us consider a trivial example:

```
10  LET X = LOG(2.39)

20  LET Y = X ↑ 3 + 5

30  PRINT Y

40  END
```

This is a complete set of instructions in BASIC. It is almost self-explanatory, one needs only point out that X^3 is written as X↑ 3, since exponents cannot be typed on a teletype. The result of the program is $[log(2.39)]^3+5$. If the student now types RUN, his program will be translated into machine-language, and executed. The answer will be printed on his teletype.

He may now wish to make a change. For example, he would like to print both X and Y. He simply types:

```
30  PRINT X,Y
```

and his old instruction 30 is replaced by the new one. He is then ready to type RUN again. The entire correction may have taken 15 seconds. If after making several corrections he is not sure of what his present program looks like, he types LIST, and receives the up-to-date version of his program.

During these operations a variety of functions are carried out by the Datanet, the 235, and the disc. His typing is received by the Datent, which stores his program and the corrections on the disc. When the instruction RUN (or LIST) is received, the Datanet commands the 235 to go to work. First the utility program is called that takes care of sorting out instructions and corrections. Then the BASIC compiler translates the program into machine language, and attempts to execute it. If answers are obtained (or error messages generated), these are written out on disc, from where they are sent to the user's teletype by the Datanet.

All of these activities take place, but the user is not aware of them! As far as he is concerned, he is talking to a single machine, and that machine talks in BASIC.

Nor is the typical user aware of the fact that he is sharing the computer with some 30 other users. The Datanet is capable of talking simultaneously to all users. For example, if there are 35 users on the system, and all are having answers typed, the Datanet will keep all 35 teletypes running at maximum speed, typing out 35 different sets of answers. When this occurs, the 235 is standing idle. The main computer, the 235, is used only when a program needs some computing to be done. Since in a typical situation the user may type for a minute, then require 2 seconds of computing time, and then his answers may take another half-a-minute to be typed, it is not surprising that the 235 can serve 35 users.

At any given time only 3 or 4 programs are likely to require service. The 235 serves the first one, places the answers on the disc, and then is ready to serve the next one. The chances are that it will have completed three jobs before the first line of output is typed.

But what if a given program requires longer service? It is first given 5 seconds by the 235. If it is not finished, all the work is written out on the disc, and all the other programs waiting for service are given a chance. Then our longer program is brought back in, and given further service in 10-second installments, until it is completed. It should be noted that 5 seconds allow the completion of 100,000 arithmetical operations! This is more than enough for most jobs.

Most requests are very short. The user makes a correction, types RUN, and again obtains an error message. This takes a fraction of a second. Or he asks to have his program listed, or he tries a simple test run. These requests are usually serviced within 10 seconds, a time just long enough for the user to catch his breath, and if one requires a substantial amount of computing time, one is prepared to wait a few minutes for the results.

An important additional service is the saving of programs. If the student has successfully debugged his program, and wishes to use it again, he simply types SAVE. His program is then placed in a special area on the disc, from which he may retrieve it the next day (or next month) by typing OLD, and identifying his program. It will then be available as if he had just typed it.

This feature enabled us to build up a significant list of library programs -- programs of general usefulness, available to all users. Thus in addition to retrieving his old programs, the student may call on a library of some 500 programs. These range from standard mathematical routines, through routines needed for specific courses, to a variety of highly popular games. The student can challenge the computer to a game of three-dimensional tic-tac-toe, or quarterback Dartmouth's football team in a highly realistic match against arch-enemy Princeton. The program is somewhat biased -- Dartmouth usually wins.

A remark is in order concerning the use of computers to play games. They are a magnificent means of recreation. But some people feel that it is frivolous to use these giants to play games. We do not share this prejudice. There is no better way of destroying fear of machines than to have the novice play a few games with the computer. We have noted this phenomenon many times, particularly with visiting alumni. And most of the games have been programmed by our students, which is an excellent way to learn programming.

We should, in this connection, note one short-coming of our equipment. While originally we were able to save user programs up to 3 months, this time has dangerously shrunk. As the library grew, and the number of our users increased significantly, the disc kept being filled up more and more rapidly, until now programs are saved for only a couple of weeks. It is clear that a larger random-access memory would be highly desirable.

Once the initial time-sharing system was in operation, our programming staff was used both to improve its efficiency and to provide further services. The language ALGOL was added for experienced programmers, and more recently also FORTRAN for those who could not be weaned from the language they first learned.

Then a large editorial package was developed. It started with programs that would rearrange the user's BASIC program, or combine two such programs into one. But it has grown into a very sophisticated system. For example, the user may now -- with a single command -- change all occurrences of 'A7' to 'B2' in his program. Or he may find all instructions that use the word 'READ'.

Another milestone was the development of "background" capability. Originally this arose out of the irony that even though 35 users were being given excellent simultaneous service, the 235 was often idle for several seconds. This time is now used to work on background problems; problems that were not initiated from a teletype. These are "spare time" tasks for the computer, which are not allowed to delay service to teletypes. But since these background problems are allowed to use all the peripherals, they enable our users to input problems from cards or tapes, to print answers on the high-speed printer, and generally to run problems too large for ordinary time-sharing treatment.

We must also mention our TEACH system, without which the freshman training program would not have been possible. We train 650 freshmen in a given year. Each of them has to debug 4 programs, and may make several false attempts before succeeding. Thus someone has to check about 10,000 programs per year. Only a machine is capable of doing this. And at Dartmouth the machine _is_ capable of doing so.

For example, one of the problems in freshman calculus, TRAP, requires the student to program numerical integration by the trapezoid rule. (See appendix IV.) We have written, once and for all, a program in BASIC that checks out TRAP programs. When the student thinks that his solution to TRAP works, he types TEST. Then the special test program takes control of the student's program and checks it out under various conditions. If the program does not work, the machine gives a hint as to what is wrong. And if the program passes all tests, a congratulatory message is typed out. The student tears off this message, hands it in to his instructor, and receives credit for the work.

We are now paying the price of our own success. Demand for more teletypes and more computing time has spread throughout the campus. We have also seen what a great service we have provided to the handful of secondary schools and colleges that have been regular users. Therefore, it was decided to launch an even larger and more ambitious time-sharing system. It is based on a GE-635, four Datanet-30's, and much larger random access memories. It has been developed jointly with the General Electric Company, and is presently about where our old system was in September 1964. When it is fully implemented, it will allow simultaneous use by 200 people, and will be much faster, more sophisticated, and more versatile than our present system. It will also allow the running of large research problems. When this stage is reached, Dartmouth hopes to provide computing service for colleges and secondary schools throughout Northern New England.

3. Impact on the Student Body.

All freshmen who complete a year of mathematics are required to complete the freshman computer program. At Dartmouth this includes about 80% of each class. The training takes place in one of two sequences: Science oriented students take a year of calculus in their freshman year, and the computer program occurs in the second semester of calculus. Liberal arts students take an introduction to the calculus followed by a course in finite mathematics. Thus their computer training is linked to the latter course.

Details of the freshman training program, including an evaluation, will be found in Appendix III. We will simply summarize some of the main features here. In either sequence the students attend two one hour lectures, and then are handed a short manual on BASIC. We have found that this very brief training is adequate. A student evaluation showed that the original 3-lecture program was too long!

After the initial lectures, each student has a teletype reserved for him for 3/4 of an hour per week, for 9 weeks. He must write four assigned programs, debug them, and pass the computer TEST on each program (see Section 1). Our experience shows that students spend between one and two hours a week on their programs (including teletype time), and that 95% of the programs are successfully completed.

In either sequence the first program is a very simple one, while the other three are substantial problems, closely connected to the course material. The last program in the calculus sequence is a simple but general purpose program for solving first order differential equations. In the finite math course the last program requires the student to estimate the limiting probabilities of a Markov chain by simulation. These are both quite demanding problems. The phenomenal completion rate testifies not only to the practicability of the freshman training program, but also to the fact that students thoroughly enjoy the work.

Any upperclassman who elects a course for which computer work is useful will almost surely have had the computer training. Therefore, instructors feel free to assign computer problems. And many students elect to do term projects on computers. There is a story circulating on the campus about a certain science course in which the instructor has for years assigned the same pet term project. This year three students handed in the project the next day -- and ten others like it. It will take a while until we fully appreciate that long laborious projects of the past are trivialities when done on the computer.

The use of the computer for instructional purposes is still somewhat haphazard. Only the engineering and business schools have thought through the implications systematically. But individual professors in all the sciences and several social sciences use the computer regularly. And students on their own have used the computer for music projects. It is clear that we still have a great deal to learn about the optimal use of machines for teaching purposes.

Of the many departments using the computer we have selected five, to give more detailed examples. They are the departments of mathematics, physics, and psychology, and the schools of engineering and business. But, before giving these illustrations, it will be useful to make two general remarks.

First of all we should like to distinguish between two entirely different ways of using computers in instruction. One is to use them as teaching machines; to have the machine teach the student. Dartmouth has done little in this area. While we ought to do more, we are somewhat skeptical about the far-reaching claims made for this use of machines, especially in higher education. The other use is to have the students program the computer; in effect, here the student is the teacher and the machine learns. We feel that this use of machines is tremendously valuable. Not only does it increase the power of each student in doing scientific problems, but there is no better way of learning an algorithm than to teach it to a computer.

Secondly, we wish to give some indication of the computing power placed at students' finger tips. The Project Director was a member of the Los Alamos Computation Center in 1945. This center represented the most powerful computing complex available just over 20 years ago. It consisted of 17 IBM bookkeeping machines. It was staffed by 15-20 employees who operated the machines 24 hours a day, six days a week. The problems solved were instrumental in designing the atomic bomb and in estimating the effect of the bomb. All the work done in a year at Los Alamos could easily be done by a Dartmouth undergraduate in one afternoon. And he could do this while 30 others are using the time-sharing system!

Mathematics

The mathematics department has prepared "teaching supplements" to illustrate the use of computing in connection with four standard courses. Manuals in Logic, Number Theory, Statistics, and Linear Algebra are available.

Each of these manuals consists of annotated programs, ranging from the elementary to the advanced. They show how a course can be enriched by combining it with computing experience. In practice, students are asked to write some of the programs for themselves, since they learn a great deal from this experience. Other programs are placed in the library, for use by students.

Let us consider the theory of numbers as an example. The Euclidean algorithm is one of the most ancient and most important algorithms in mathematics. It finds the greatest common divisor (g.c.d.) of two numbers. It is amazingly simple, very fast, and ideally suited for computer work. But the student misses the simplicity when he gets bogged down in arithmetical computations. And he fails to realize the power, since large examples take too long to work by hand.

```
LIST


EUCLID    21:50   MAR.27,1967

10 PRINT " A", " B", "G.C.D."
20 READ A,B
30 PRINT A,B,
40 LET Q = INT(A/B)
45 LET R = A - Q*B
50 LET A = B
55 LET B = R
60 IF R > 0 THEN 40
70 PRINT A
80 GOTO 20
90 DATA 130,169, 243,256, 123456789,987654321
99 END

RUN


EUCLID    21:51   MAR.27,1967

 A               B               G.C.D.
 130             169              13
 243             256              1
 123456789       987654321        9

OUT OF DATA IN  20


TIME:   0 SECS.
```

We show a listing and a run for the program EUCLID, which carries out the algorithm. The program is amazingly short. The whole algorithm is contained in the five instructions on lines 40-60; the rest of the program reads data and prints answers. The sample run finds g.c.d.'s for three pairs of numbers, the last being in the hundreds of millions range. Computing time is rounded to the nearest second, so that 'TIME: 0 SECS.' shows that the translation of the program from BASIC to machine language plus the execution took less than 1/2 second.

Equally impressive is the traditional sieve for finding primes. It is hard to improve on it -- but students delight in doing so. Much more ambitious is the finding of large twin primes (see the next section for an example) or programming the Chinese remainder theorem.

In logic the computation of truth tables or the checking of whether a formula is well-formed are natural candidates for computer programs. In linear algebra all kinds of work with matrices is facilitated by computers, and the simplex method for linear programming was designed for machines. But the best example is statistics. Anyone who reads the statistics teaching supplement will wonder how statistics was ever taught without having a computer at each student's disposal.

We strongly urge the interested reader to send away for one or more of these teaching supplements. They are our evidence that computers do enrich the undergraduate mathematics curriculum, and that they make courses vastly more interesting.

Engineering (Thayer School)

The time-sharing system has become an integral part of our educational and research activities. The time-sharing man-machine interaction has produced a situation in which staff and students can implement their concepts by programming their mathematical models, correct their programs and obtain answers 10 to 100 times faster than a person operating in the most efficient batch-processing system.

Introductory programming using BASIC language is taught to all of our freshmen in their mathematics course. The students then find it convenient to use the computer for their homework and term assignments. They use the computer to solve course problems without being told that they should do so. Students naturally try to do things the easy way. The time-shared computer enables them to do homework faster and easier than by working longhand. By the time the student has reached his junior year, he is a steady programmer, often working two or three digital programs per week as part of his regular course work.

Our students have developed an attitude that we believe will become characteristic of engineers in the future. Our students consider that getting a numerical answer is secondary to the matter of problem formulation. They are willing to consider the analysis of much more complicated systems than heretofore and have no hesitation in exploring the behavior of a complex mathematical representation.

Fully half the students in the courses in fluid mechanics and solid mechanics write programs for data reduction and analysis of their laboratory work. We have found a much closer connection between what is done in the laboratory and the mathematical modeling because one can go back and forth so easily.

The impact on design education has been particularly helpful. Sophomore students in their first engineering course "Introduction to Design" wrote and used a BASIC program which related torque, angular position, velocity and acceleration for the rotation of a human arm in raising itself and a weight in a vertical plane. They used these design calculations in their project work.

At the graduate level the effect is equally dramatic. We now ask students to prepare a mathematical model for an entire processing plant. They do this work within three or four weeks time. Such an achievement is only possible in a time-shared environment.

Last February an example problem was needed for a workshop session for a course in digital simulation for practicing engineers in Sarnia, Ontario. The course used the Dartmouth computer in Hanover, New Hampshire. A quick interrogation of the computer files produced a seven stage extraction problem. Dan Frantz, a first year graduate student, was asked to help. Dan was a physics major at Michigan. After a 20 minute description of how an extractor works, Dan went to his desk and wrote his computer unit calculations in one hour. An hour and a half later he had programmed the unit calculation, fitted it into an executive system for this kind of problem and in three hours, by the clock, he had returned with the answers. The program was then used via long distance in the demonstration at Sarnia, Ontario.

In a graduate level course in process simulation and design, three graduate students were asked to simulate a complete process plant in a six week period. All were familiar with BASIC but it was necessary for them to learn ALGOL, become familiar with the time-sharing version of PACER (a process oriented language) with the mathematical models and to simulate their plants. The mathematical models developed were limited in detail because of memory restrictions but they do essentially all of the heat and material balances for the entire plant. These mathematical models provide a basis for more comprehensive and larger simulations. The total cost in computer time was approximately $200.

Another student, as part of his design project, used the computer to generate a table for morphological synthesis. He produced over 5,000 alternative system concepts from function tables entered into the program. The resulting listing was very helpful because it forced the student to consider the potential design success of unusual system combinations. It required only three hours for programming and typing and an additional hour to obtain the results. This sort of work is helpful because of the iterative nature of the design process. It is extremely important to students to be able to generate and evaluate design alternatives quickly. With time-sharing it is easy for students to modify systems and generate many cases in a term course.

In a graduate course devoted to decision theory, before time-sharing the instructor used to assign two problems for an evening's homework. With the time-shared computer available, the assignment was changed. Students were asked to develop a computer program that would solve 15 problems listed in the text and to present all 15 solutions. The students not only program the computer to solve these problems in one night, they also submit an analysis of the class of problems which can be handled by the methods given in the book. The difference in student understanding is truly amazing when one compares their performance after they have programmed a computer to solve a problem as compared to merely solving the problem.

The examples given in this report represent a small sampling of the activities at Thayer School. The computer is now such an important tool in engineering education that we would be totally lost without it. Our students decide whether or not to go to another graduate school on the basis of whether they will have access to a similar time-shared computer. We know of several students who have gone to another institution for doctoral work and immediately made arrangements through research contracts to have a teletype available and tied into a Dartmouth-type time-shared system. Since the Thayer School of Engineering will not permit its own students to go on to the doctorate at Dartmouth College, but insist that they go elsewhere, we have been under considerable pressure to allow our students to continue with us simply because, at the moment, there is no place else in the United States where they can get such good computer service.

Psychology

Three points will serve to highlight the impact of BASIC and time-shared computing in general on the activities of the psychology department. (a) The ease with which BASIC can be learned has given rise to a requirement that all psychology majors (as of Fall 1965) shall know how to program in BASIC. No credit is given for this ability since the use of time-sharing computers is considered to be the accepted way of analyzing all manner of data.

(b) The psychology department is proud of its experimental course sequence - Psych 61-62-63 - in which students perform experiments, obtain data, and apply statistical tests on either their own data or that for the group. One of the practical problems of analyzing group data was the simple requirement that all the data for all students should be available to all members of the class. On several occasions this was accomplished by assigning a DATA instruction number to each student, asking him to call in the stored program, enter his data in the appropriate instruction number, and then SAVE. By the end of the week most all of the data was thus stored in the program and the analysis could be RUN. In a similar vein, the instructor was able to gather group data in a class session, have an assistant type in the data in an already available program, and have the analysis available for discussion with the class - all within the 50 minute period. This procedure was most useful in connection with demonstrations of some standard scaling procedures (paired comparison scaling, direct magnitude estimation scales, balanced incomplete block designs, and so on).

(c) Perhaps the most important point that can be made is this: As students learn to operate on a high-speed computer several attitude changes are evident: (i) the fear and superstition regarding "automation" evaporates; (ii) social science students, traditionally apprehensive of things mathematical and statistical, begin to feel a power over these things; (iii) students realize that ability to use a computer opens up many opportunities in the way of thesis topics and, later, in the kind of career they plan; and (iv) students communicate with each other more once they learn to use the computer - they feel a sense of camaradie with other users. Nowhere has all this been more evident than in the statistics course offered in the psychology department. Both from the instructor's point of view and from the students' point of view a statistics course taught with the aid of a time-sharing computer is an exciting experience. No longer do coins or dice have to be tossed 10,000 times - using the random number generator enables samples this size to be simulated in seconds; no longer is sampling from a normal distribution practically impossible - algorithms are available for doing this in seconds; no longer do sampling distributions of statistics stay in the background - students can plot the sampling distributions, can approximate the areas under these distributions, indeed, can make up their own statistical tables. A great deal more is accomplished in an introductory statistics course under this system and the feedback from the students is most encouraging.

The success of this statistics course has led to the writing of a new textbook: STATISTICS - TRADITIONAL AND BAYESIAN*, in which exercises are geared essentially to the use of a freely available time-sharing system.

---

* by Victor E. McGee for Appleton-Century-Crofts (early 1968).

Physics

The most significant application of computers in physics education has been the following: A sequence of five programs has been developed to assist the student to explore in depth several features of the bound states of a one-dimensional square-well potential, both in the coordinate and the momentum representations. The square-well problem was chosen because the analytic solution is well within the grasp of the beginner in quantum mechanics, thus permitting an inter-play between analytical and numerical methods. This project was carried out with great success by 80 second-year students at Dartmouth College, (See the report in American Journal of Physics 35, 275 (1967).)

The computer is employed to assist the teaching of an optics lab, where exact calculations are tedious and uninteresting. Students also turn to the computer as a matter of course in order to carry out such tasks as experimental curve-fitting, solution of differential equations and other numerical integrations, Monte Carlo calculations, and routine arithmetic manipulations formerly consigned to the desk calculator. Graduate students find it a vital research tool, as do some faculty members, who also depend on it to process grades in the high-enrollment courses.

Projected use of the computer in this department is expected to increase and at an increasing rate. The new capabilities suggest new uses and stronger demands. Under discussion is a freshman lab organized around the computer is such a way that the physical phenomena of mechanics might be displayed in tight conjunction with the mathematical models intended to describe them. This would have the advantage of permitting the student to "play" with the parameters of the theory, much in the same way that he plays with experimental parameters in a conventional lab.

Business Administration (Tuck School)

The system has received extensive use by students and faculty, in classroom applications, extra-course professional uses and personal uses. A measure of the growth and extent of usage at Tuck is the number of teletypes located at the School: one in spring of 1964, 2 during 1964-65, 4 during 1965-66, and 8 during 1966-67. A new system, LAFFF (Language for the Aid of Financial Fact Finders) was conceived, developed and implemented by Tuck School personnel (See below). A Tuck School Associates program was held in June of 1966 which featured the time-sharing system in financial analysis. In addition a number of formal demonstrations of the time-sharing system were given by Tuck School personnel in Hanover and elsewhere including a three-session series to Dartmouth alumni in Washington, D.C., Philadelphia and New York in recent months.

LAFFF. The LAFFF system was conceived as an aid to those involved in analyzing data on publicly traded companies. It enables users of the Dartmouth Time-Sharing System to retrieve and manipulate financial information from a data bank drawn from the Standard Statistics Corporation's COMPUSTAT tapes. It is the first such operating system. Currently the data bank contains thirty financial facts (e.g. price-earnings ratio, dividends, closing price, etc.) for each of thirty-six companies, for each of the last ten years. Manipulation includes addition, subtraction, multiplication, division, correlation and growth rate. The next version of the language (currently under development for the GE 635 system) will provide fifty facts for some 900 companies for the last twenty years.

A description and discussion of the LAFFF system can be found in "A Language for the Aid of Financial Fact Finders" by R. S. Bower, C. E. Nugent, J. P. Williamson and B. C. Myers, Financial Analysts Journal, January-February 1967.

The LAFFF system has been used extensively by Tuck School students for required course work and projects and by faculty members in connection with their research (see below).

Course Work. The strategy at Tuck School is to expose every incoming student to the Dartmouth Time-Sharing System as part of a required course "Electronic Computers and Their Uses" which starts on the opening day of the students' two year tenure, and then to capitalize on and augment that required exposure by judicious use of computer-oriented assignments throughout the curriculum.

As part of the required computer course, each student is obliged to program the solution to about a half dozen business oriented, increasingly difficult problems. The problems are graded by the computer. These provide merely a grounding and a starting point for his computer-oriented experience at Tuck. Discussio of a few examples follows.

Accounting and Finance. An important assignment is to obtain company financial facts through LAFFF, reconcile financial statements obtained from the LAFFF data with company financial statements as printed in their annual reports, and prepare probabilistic projections of financial statements for the next five years using a given behavioral model. This assignment teaches use of the LAFFF and BASIC system, promotes familiarity with the make-up of the financial statements and provides an appreciation of the power of Monte Carlo simulation based on a behavioral model. Later in the course the students are asked to improve the model and use the projections in the role of management, a banker faced with solvency questions or an investment broker.

This course also uses the time-sharing system for capital budgeting (rate of return, present value, compounding periods, annuities, etc.), cost of capital using LAFFF and BASIC, and lease bargaining.

Production. The time-sharing system is frequently used in connection with this course. The most notable use is in a production-inventory simulation exercise called UNIPRODUCT.

Briefly, in this exercise the students evolve good production planning rules by testing prospective rules on a manufacturing simulation routine containing production uncertainties.

Managerial Economics. This is basically a course in model building laid in the economic context. The heaviest use of the system in this course is in the building and testing of regression models as well as study of the regression technique itself. In the study of regression, Monte Carlo analysis provides a graphic illustration of the nature of the statistical estimates and also of the problems of regression (serial correlation, heteroscedasticity, auto--correlation, etc.). The system is also heavily used for student projects in this course.

Quantitative Analysis. Among the uses to which the system has been put in this course are: generation of probabilities from standard distributions (binomial, hypergeometric) for values not tabulated, solution of linear programming problems and exploration of the sensitivity of the solutions to selected inputs, better understanding of probability distributions through the Monte Carlo generation of observations from a range of distributions (Normal, Binomial, Exponential, Poisson, Gamma), better understanding of the Central Limit Theorem and the Law of Large Numbers through their graphic illustration via Monte Carlo methods, better understanding of hypothesis testing by the Monte Carlo production of sampling distributions.

Topics in Operation's Research. Typical uses are: Many applications in mathematical programming, queueing theory, regression, game theory and simulation. Several computer-oriented course projects include ones in Job Shop Scheduling and in Marketing Research.

Investments. Many uses in portfolio selection, use of statistical indicators, bond bidding, measures of mutual fund performance. Student projects were done with computer-oriented study in the areas of bond switching, call premiums, technical analysis, etc. The final examination this year included a question which required the students to go to the teletypes and test a certain growth hypothesis using LAFFF data.

4. Impact on the Faculty.

The previous section gave many illustrations of the usefulness of the time-sharing system for the college teacher. We will now consider its impact on faculty research.

We claim that a time-sharing system increases the number of faculty research uses of computers by an order of magnitude. For every large research project that makes the use of computers inevitable, there are ten projects for which computers are useful if they are simply and quickly available, but which do not justify going through the delay and inconvenience of a "batch-processing" system.

An important research use of the machine is to compute examples. Both in mathematics and in various branches of science theoretical work requires the computation of carefully chosen examples. One normally works out two or three examples, at a considerable expense in human labor, and with a significant chance of making a mistake. Any example that is at all possible without a computer is a triviality for the machine. Thus at the cost of writing a simple program one can obtain dozens of examples, and obtain them in a few minutes. The same examples would be useless if one had to wait several days until the program was debugged. When one has a hot streak in research one wants examples right then and there, or not at all.

Equally important is the analysis of laboratory data or observations of natural phenomena. The researcher can write a program once and for all, save it, and use it each time new data is collected. He simply calls up the old program, types in the data, types RUN, and the answers appear.

But we must not give the impression that the time-sharing system is useful only for small research problems. We include a sample of a major research task carried out by the Project Director. It is in the field of the Theory of Numbers, and concerns the search for large twin primes. Prime numbers are the building stones out of which integers may be built by multiplication; primes cannot be further decomposed. The first few primes are 2, 3, 5, 7, 11, 13, and 19. All but the first are odd. When two consecutive odd numbers are primes, such as 3 and 5, or 11 and 13, we call them "twin primes". While Euclid already knew that there were infinitely many primes, we still do not know whether there are infinitely many twin primes. Therefore, many mathematicians have searched for large twin primes.

The Project Director happened to come across two different references to the "largest known twin primes", which were slightly beyond 10^{12} (a trillion). He was interested in seeing whether the time-sharing system was powerful enough to find a larger pair. He asked the computer to search the two thousand numbers starting at $10^{12} + 10^6$ for twin primes.

The computer found two such pairs:

$$10^{12} + 10^{6} + \quad 1341 \text{ and } 1343$$

$$1941 \text{ and } 1943$$

These were at that time "the largest known twin primes".

Several remarks are in order. First of all, the total time to find these results was 1 1/4 hours. During this time less than 11 minutes of computing time was used, and there were 20 other people using the system! Once the problem was entered, the Project Director went on with other research, simply waiting until answers were typed out. Any mathematician would agree that this is a very substantial computation, and it is easy to do it in a time-sharing environment. But it also shows that there is little point in finding larger pairs of twin primes. Anyone with a similar system at his disposal can, in one evening, beat the record.

```
HUGEPR    21:00   18 ØCT. 1965

PRIMES ØF THE FØRM  A*1000 + B,    0 < B < 2000.

A = 1000001000


VALUES ØF B YIELDING PRIMES:


 21     59     69     93     99     107    111    129    197    213    231
 249    263    321    333    347    353    399    491    513    549    561
 569    591    639    693    711    731    737    749    759    773    797
 819    941    1023       1059       1091       1107       1113       1119
 1127       1163       1169       1187       1193       1239       1301       1341
 1343       1383       1403       1409       1413       1431       1443       1473
 1479       1497       1529       1541       1569       1577       1613       1691
 1911       1931       1941       1943       1953

THERE ARE  70   PRIMES IN THE RANGE.


TWIN PRIMES:

 1341      1343
 1941      1943


TIME:  10 MINS. 32 SECS.



USE

 21 USERS AT 22:15.
```

But the most important development has been what we learned about man-machine interaction. There have been many debates as to what tasks are most efficiently done by human beings, and what tasks should be left to computers. We now feel that in most cases both solutions are wrong: The tasks should be done by man-machine teams. This is really efficient only in a time-sharing environment.

Let us consider a complex problem, which may involve millions of computations. The programmer must foresee everything that could possibly happen during the logical chain, and provide for it in his program. Human beings are not very good at doing this. Or if they succeed, the price is a tremendously complex program that must provide for all kinds of catastrophes, most of which never occur. In a time-sharing system there is a much better procedure. Let the machine do a reasonable piece of the work, and then report the results to the programmer. He can then use his judgment as to whether to proceed, or to change the plan of operation. It is thus not necessary to program human common sense -- it can be applied by the human being when needed.

This is the most important lesson we learned during the project, and we did not predict it at the outset. Many of us who had significant previous computing experience have changed our entire approach to computers. We have learned how to work with the computer in solving a problem, rather than submitting a problem for machine solution.

We will illustrate this with a dramatic example. One of our colleagues spent a term at another institution, which has an excellent "batch-processing" computer center. He was working on a model in Sociology. He had conjectured that a probabilistic process tends to a limiting distribution, which should be quite evident after 50 experiments. Many dozens of 50-experiment sequences were simulated on the computer, but they did not suggest a reasonable distribution. He returned to Dartmouth, where he decided to continue the computer experimentation, but this time he was doing it himself, sitting at a teletype. When a particularly unlikely series of experiments came up, he asked the computer to continue. The next 50 experiments changed the answer drastically, so he continued even further. Eventually he discovered that his hypothesis was incorrect, 50 experiments give no useful information. In the long run all series of experiments converge to a fixed outcome. He might never have made this discovery without a computer at his finger-tips. And in the meantime he would have wasted an enormous amount of computing time.

We have several instances of faculty members who have obtained new results, or corrected old ones, due to the availability of time-sharing. These were usually in fields where the problems were considered "too small" to justify the use of a high-speed computer. And yet the problems were much too difficult to do without a computer. The time-sharing system closed the gap.

5. Impact on Other Institutions.

Although this project was directed mainly at improving the teaching of mathematics and science courses at the college level, part of the project involved placing a teletype machine in the Hanover High School. One purpose was to learn how effective easy-access computing could be as an aid to secondary education. Another purpose was to learn how far down into the secondary elementary grades computing could be effectively handled by the students.

At the same time, a number of input ports from the computer were attached to the Hanover phone system so that teletypes located away from the campus could dial the computer. This permitted several public demonstrations in distant cities (including Edinburgh, Scotland) and permitted other colleges to experiment with the type of computing being supplied to the Dartmouth campus.

There are now about a dozen input ports on the computer available to the outside world. Current users include schools, colleges, government agencies, and some local business concerns. The rapid acceptance of this type of computing, primarily in the schools, leads us to assert that time-sharing is the sensible way to supply computing power to the secondary schools. First of all, there is little monetary investment on the part of the schools, and they can cancel their teletype service on short notice. Second, the installation of a teletype machine requires far less than the installation of a small computer, and much less administrative and technical support as well. Finally, the user has available through the teletype a much wider variety of services and sophisticated languages than could be provided by a small, free-standing computer.

Use by Colleges

In the early days of the project a number of colleges and universities experimented with the Dartmouth System over long distance telephone lines. In most cases the purpose was to familiarize themselves with the type of service available from a time-sharing system. In some cases use by students in courses occurred. A few used the teletype machine for significant amounts of research work, notwithstanding the presence of a conventional batch processing system on their own campuses. More recently, a small number of colleges have installed a teletype machine for large-scale training and use by their students.

The use by colleges has been limited by the small number of available telephone lines into the computer. However, the total number of schools is large, and their distribution impressive.

Most of them are listed below:

Princeton School of Engineering
University of Michigan (3 departments)
Harvard - Business School
Harvard - School of Education
Harvard - Statistics Department
State University of New York at Binghampton (Harpur College)
  Plattsburgh
  Stony Brook
Mount Holyoke
Smith College
Middlebury College
Amherst College
Williams College
NYU Medical Center
NYU Computation Center
Ohio State University
Standford University
Rennsalear Polytechnic Institute
Brooklyn Polytechnic Institute
Carnegie Institute of Technology
Johns Hopkins University
Lehigh University
University of Maryland
McGill University
University of Pennsylvania

In some cases Dartmouth has contributed to this usage through an educational grants program.

The number of colleges interested has grown so rapidly that we have been encouraged to establish a Regional Computation Center organization to better serve their interests and to learn what problems and costs are involved. The notion of several schools sharing a large computer for educational purposes has recently been given strong encouragement by the report of a Panel of the President's Scientific Advisory Committee entitled Computers in Higher. Education. This report of the so-called Pierce Panel was in turn strongly influenced by the results achieved on the Dartmouth campus under the auspices of this project.

Use by Secondary Schools

Perhaps the most startling of the unexpected effects of this project has been its rapid adoption into secondary schools. This started in the fall of 1964 with the placement of a teletype machine into the local Hanover High School, mainly to see what would happen. On the basis of this experience we now feel that computing will soon become a necessary part of the secondary curriculum. We also feel that it may be appropriate to introduce computing to the students as early as seventh grade.

While much work still remains to be done in terms of curriculum materials and teaching aids, the amazing success so far with little more in most cases than the standard BASIC Manual strongly suggests that only moderate curriculum development and teacher retraining programs will be needed.

One method of introducing students to the computer is through a Computer Club. One was established at the Hanover High School as an extra-curricular activity in the fall of 1964. During the next school year several hundred students became exposed to computing, and some of them became very proficient. One of the startling resluts was the work done by a small group of talented fifth-graders. Learning from the BASIC Manual without formal instruction, they produced some quite sophisticated programs: one, an improvement on a library program to factor integers; and another, a program to generate magic squares. They quickly absorbed the other languages available in the system, which attests to the great efficiency and usefulness of time-sharing for learning programming.

Some of the other work done by Hanover students included a program for scoring a debate tournament, and another for playing a game of chess. The latter cannot compare with the chess-playing programs existing at MIT and Stanford, but this program exists within the much stricter limitations imposed on program size by our system.

At the present time the seventh grade mathematics instructors are experimenting with teaching BASIC as a regular part of the mathematics course. The obvious advantage for computer instruction at this level is that it can be used in all subsequent mathematics and science courses. Experience to date shows that this training can be done at this level with average students, but that instructional materials appropriate for this level need to be produced. Seventh grade students are capable of producing quite complicated programs as long as high school or college mathematical topics are not required.

The second school to install a teletype was the Phillips Academy at Exeter, New Hampshire, in the spring of 1965. Installed primarily for a summer session, it quickly became a permanent fixture. Being a boarding school, the teletype was used during evening hours as well as during the day. Some students even obtained permission to get up before "reville" to make fuller use of their teletype machine. Here as at Hanover several hundred students obtained instruction in computing, and used it in their courses.

Since then many other schools have inquired about "hooking up" to our system, and we have been able to accomodate a few, which are listed below:

Hanover High School
Phillips Exeter Academy
Phillips Andover Academy
Mascoma Valley Regional High School
Mount Hermon School
Vermont Academy
Kimball Union Academy
The Holderness School
St. Paul's School

The work at several of these schools deserves special mention.

At Mascoma Valley Regional School, a small school serving a rural area, the use of computers by the students was received so enthusiastically that within one year the School Board was able to convince the School District to appropriate funds for the rental of a teletype machine and long distance phone line. The mathematics instructor in charge, Mr. Richard Moulton, also conducted adult education classes in computing in the evenings. He was recently commended by the New Hampshire Association for Better Schools for his outstanding work in computing with the Mascoma students. This was all done with no more assistance on the part of Dartmouth College than the supplying of BASIC Manuals and the answering of occasional questions.

The St. Paul's School in Concord, New Hampshire, made heavy use of the computer via an installed teletype machine for their summer school program. Computing was included as a part of the Concepts in Mathematics course. The program was so well received that the School elected to continue the program into the regular school year.

Computer use at the Mount Hermon School in Mt. Hermon, Massachusetts, began only in the fall of 1966, but in one semester they trained 400 students (one-half of the student body) on a single teletype machine. They now plan to indoctrinate each incoming ninth grade class in computing.

For the computer usage described above, Dartmouth has contributed heavily through a program of education grants. Many schools were thus able to begin work quickly. This program is especially important to public tax-supported schools which would otherwise need two or three years to develop their own funds.

Dartmouth is now in the process of organizing a project for collecting and documenting the exercises and techniques found useful at these schools, and for trying them out at other schools. If this project comes about, the materials produced will be available to any other school in any part of the country wishing to integrate computing into its educational program. In any case, the number of secondary schools using the Dartmouth computer is expected to increase to about 20.

An Educational Utility

More than any other single project, the success of the Dartmouth project has made concrete the ideas and usefulness of an Educational Computing Utility. It has been shown not only to be possible but also to be in great demand by the schools and colleges not already having their own computers (and by some who do!) We have shown that the problems are not great, and that the cost, though substantial, is not out of the reach of most secondary schools and colleges, even with the relatively crude system first used by us.

With our new system, which will be capable of supporting many more users simultaneously, we estimate that the costs for normal use will be around $5000 per year full-time teletype machine.

Our experience does not include what is usually called CAI. While more elaborate CAI systems using fancy terminals (including TV-type presentations) could be attached to a general purpose computer system such as ours, the terminal costs would be much greater than those for teletype machines. The type of activity we are experiencing is more like that of a student teaching the computer rather than vice versa. By being able to program certain processes, the student necessarily shows a thorough understanding of the process. For example, one student in sixth grade was able to write a program to add fractions, thereby showing a complete understanding of the process. Of course, this does not develop manual skill for performing such calculations. If this is desired, the computer can easily be used to drill the student.

We have seen the computerized arithmetic drill provided by the Suppes project at Stanford. In almost all detail (except the instant response to individual characters) the automated arithmetic drill routines can be programmed in BASIC on our computer. The only difference is that in our system action is taken only upon receiving the "carriage return" key, whereas in the Stanford system action can be instigated upon receiving the first incorrect character of a wrong answer. Of course, if only automated drill were needed, a general purpose time-sharing system such as ours would not be needed, and when used in this way implies a significant amount of system overhead. On the other hand, being able to program these exercises in a simple language such as BASIC represents a real saving in program preparation effort.

The strong advantage of a general purpose time-sharing system at this time is that it can support most or all of the applications that might arise in educational work. This would include both instructional use and administrative work. The particular applications do not have to be planned in detail in advance, and the schools and colleges themselves can participate in the development of the programs and systems for their use. We feel that this is a significant advantage over a firm or organixation preparing a specialized, non-changeable education computer system and presenting it as a fait accompli to schools and colleges.

Commercial Use

The Dartmouth system has had a fantastic influence in business, engineering, and industry. While only a small number of firms have actually experimented with our system, the ideas have spread to a large number of firms. The General Electric Company is providing time-sharing service to many hundreds of customers using computer systems which are patterned directly on ours. Professional engineers and bankers alike are finding that a teletype in their office and connected with a distant time-sharing system can be extremely valuable to them.

The language BASIC has been or is being implemented on a number of different computers of different make. While Dartmouth has no direct interest in these activities, this paragraph is included to indicate the very wide influence this work has had in a very short time.

Future Plans

Plans now being implemented involve the development of a large-scale, general purpose time-sharing system capable of handling many more users than the original system. It will also be capable of internal file handling, which will make possible experiments in administrative and guidance work, and in instructional work where the use by students are not isolated experiences. It will also permit the development of on-line library service, the planning for which has already begun.

Along with this new system, which is described briefly in Appendix I, plans are being developed for about 20 secondary schools and 12 colleges to be regular users of the Dartmouth system starting in the fall of 1967. A specific purpose of the work with schools will be to develop instructional materials. A purpose of the work with colleges is to develop the notion of the regional computation network, and to determine the best ways for its administration. Both programs have as their main prupose the providing of a general purpose computing capability to these schools and colleges on a day-to-day basis.

APPENDIX I

Description of the Dartmouth Time-Sharing System

The Original System

The original Dartmouth Time-Sharing system was built around standard hardware components from the General Electric Company. All of the components were on-shelf items for which specifications existed before they were approached by Dartmouth. After the equipment was obtained*, several minor modifications were made in order to improve its efficiency. But it is important to note that specially designed hardware was not required for the Dartmouth project. Rather, it was our purpose to use standard hardware and to provide through software the desired functional characteristics.

The hardware consisted of the following items:

1	GE-235 Central Processor, 16,384 words of 20 bit memory.
1	Datanet-30 Communications Processor, 16,384 words of 18 bit memory.
1	18,000,000 charcter Disc with a controller that can be accessed by either the GE-235 or the Datanet-30.
4	Magnetic Tapes operating with the GE-235.
1	900 line per minute Printer, attached to the GE-235.
1	Card Reader, 400 cards per minute.
1	Card Punch, 100 cards per minute.

As can be seen, the capability of the above computer system to operate in a conventional batch-processing mode is hampered by the slow speed of the card reader and punch, and by the presence of only four tape drives. It should also be noted that we obtained no off-line computer equipment for card-to-tape or tape-to-printer operations.

The modifications made to the system after it arrived were these:

> Four magnetic tapes were added to improve the ability to operate with conventional batch systems but as background to Time-Sharing.
>
> The Disc Controller was modified to permit reading or writing 18,432 characters with one command, thus saving the equivalent of seven latency times for large data transfers. This modification would have been desirable in any case.

---

* with the assistance of a $300,000 grant from the National Science Foundation.

The Central Processor was slightly modified so that certain illegal instructions which originally caused a hardware halt became no-operations. This change was essential for unattended operation.

The schematic plan of the hardware system is shown in figure I-A.

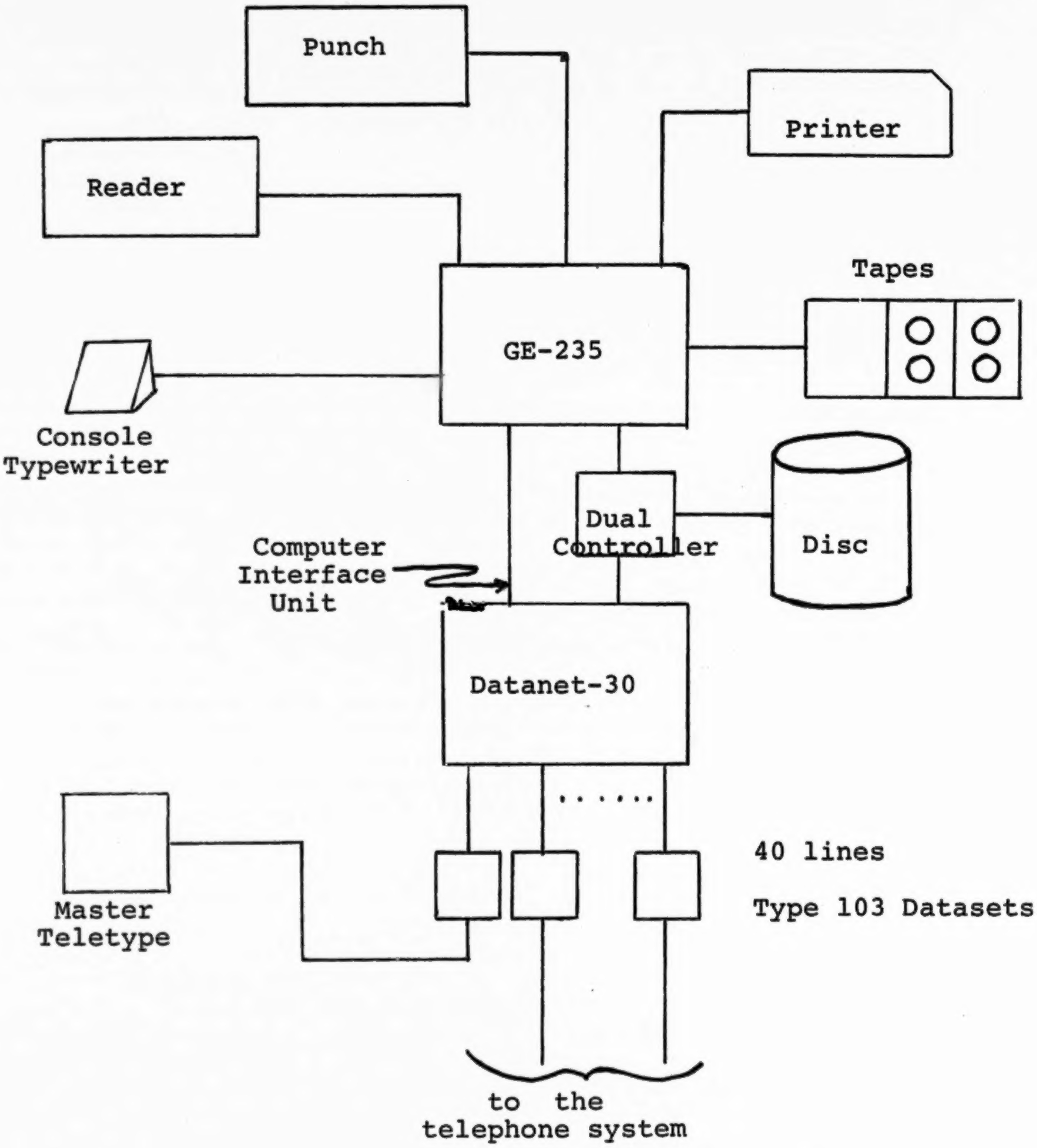

Punch
Printer
Reader
Tapes
GE-235
Console Typewriter
Dual Controller
Disc
Computer Interface Unit
Datanet-30
40 lines
Type 103 Datasets
Master Teletype
to the telephone system

Figure I-A. Schematic diagram of the hardware for the original Dartmouth Time-Sharing System. This particular configuration is known as the GE-265.

The memory of the GE-235 is accessed through a Controller Selector Unit that serves to channel requests for memory access from the various peripherals. Input and Output to several devices can thus be carried on simultaneously with processing. Upon completion, input and output operations interrupt the Processor which can then initiate further input or output. In particular, the Datanet-30 can interrupt the GE-235 processor, and can thus act as a Master computer. In short, the standard features of the GE-235 and Datanet-30 combination (interrupt-controlled and buffered input-output, dual access disc controller, computer interface unit, and the ability of the Datanet-30 to connect directly to telephone datasets) made possible a simple hardware system that could serve as a basis for a time-sharing system.

The decision was made to define the Datanet-30 as the Master computer, and equip it to handle all communications lines. It contains the functions of command analysis, allocation of disc storage space, scheduling, and control of the GE-235. The latter computer in turn handles all compiling and running, and also serves as a buffer for transferring programs from one area on the disc to another. The GE-235 is equipped with a small executive system that responds to interrupts including those from the Master Datanet-30 computer, carries out the input and output operations requested of it, and transfers control to a user program It can also serve to carry out card-to-printer, tape-to-printer, and other peripheral-to-peripheral operations without interfering with time sharing. This last capability is what permitted us to operate without any off-line punched card equipment of any sort except key punches.

The compilers are designed to be core-resident, and to compile core-to-core. While this limits the program size that can be handled in some cases, it permits unusually fast compilations. It was our decision to always recompile completely each time a program was rerun or modified for a second run. In no cases do we save the so-called binary program. We have found that a very fast compiler makes this mode of operation not only feasible but also highly desirable. The compilers are reentrant so that a new copy does not have to be brought into memory for a new job. The compiling data are retained in the core memory area assigned to the user, and, in case a swap between users is called for by the Datanet-30, the entire user core area is written on the disc. Later when it is brought back for a second turn, all the compiling data and the other necessary information is also brought back in order to continue the compiling.

Because of the severe core memory limitations in the GE-235, only one user job exists in memory at a time. Furthermore, in case of swapping, the GE-235 processor actually must wait while one job is written out on the disc and then another brought into core memory for its turn at processing. Despite this significant reduction in machine cycles that can be allocated to the user, excellent service is obtained. We feel that there are two factors that explain why: first, even when there is a heavy load, there may be only a few users actually competing for compute time. The others will be either watching their teletype produce output (which involves only the disc and the Datanet-30.) Second, we have found that the number of swaps required can be greatly reduced by granting each job an initial allocation of processing time to provide for completion in the majority of cases. For instance, with a small student job, most runs actually terminate because of **errors** detected during compilation, and the complete run might take no more than one second. Even if the program is correct, experience has shown that a large majority complete their calculations within five seconds. On the basis of experience, we have set about five seconds for the initial run of a particular job. During heavy periods this decision insures that up to 70 percent of all processor cycles are returned to the users and only about 30 percent lost in waiting for swapping and in other overhead.

The master control program resides in the Datanet-30 computer. It is thus completely protected from runaway programs in the GE-235. In fact, it is possible for the Datanet-30 to insert into the GE-235 memory a recovery program that brings in from the disc a clean copy of its executive system. (It should be noted that having only one user job in the GE-235 memory at a time protects against one user program destroying another, since hardware memory protection is not available on the GE-235 memory.) Thus, the most important reason for memory protection -- that of isolating the master executive from poorly debugged user programs -- is automatically provided for in this two computer system.

The master program in the Datanet-30 scans on a clocked basis the bit buffers associated with the teletype lines, at a clocked rate of around 110 times a second. Completed characters are handled also in "real time." In the gaps between these periods of real time servicing of the bit buffers, the Datanet-30 carries out spare time tasks. These include all input and output to the disc. In addition. the GE-235 is regularly queried to see if the previous job has been completed, or if its allotted time is up. If so, the Datanet-30 sends to the GE-235 a short message over the computer interface unit giving it information concerning what disc operations to carry out. The Datanet-30 then sends an interrupt to the GE-235, whereupon the latter computer processes the interrupt and carries out the requested disc operations.

Functions and Services

The main requirement in the design of the time-sharing system was to keep the user interface simple. The system was to be used by large numbers of persons with little classroom or formal training. The commands were designed to be easy to learn and easy to remember. The ordinary user was not required to do extra tasks for which he did not know the reason. This meant that more sophisticated users had to go through more elaborate sequences to specify their desires to the system. But the vast majority of users use only a very few of the commands.

In preparing a program, the essential features, after logging in to the computer, are (1) composing the program in a language, usually BASIC, and (2) running it. There is precisely one command for composing a new program, and that is NEW. There is also precisely one command for running a program, and that is RUN. We give an example of a user running a simple problem he has just composed.

User types underlined words.

```
NEW
NEW PRØBLEM NAME--TABLE
READY.

100  PRINT "X", "LØG(X)", "X-SQUARED"
110  FØR X = 1 TØ 2 STEP .1
120     LET L = LØG(X)
130     LET S = X↑2
140     PRINT X, L, S
150  NEXT X
160  END
RUN
```

User types this, including RUN

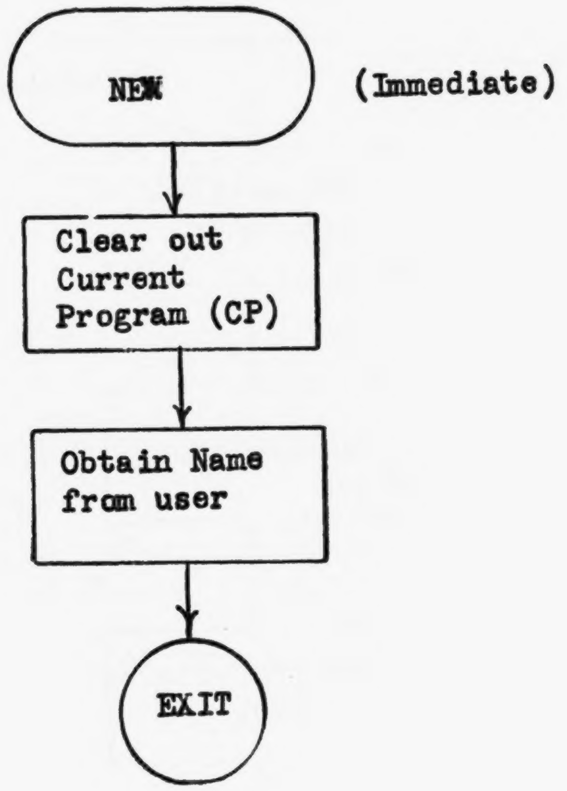
```
TABLE      17:14   MAY 19,1967 FRI

X              LØG[X]         X-SQUARED
 1             0               1
 1.1           9.53102 E-2     1.21
 1.2           .182322         1.44
 1.3           .262364         1.69
 1.4           .336472         1.96
 1.5           .405465         2.25
 1.6           .470004         2.56
 1.7           .530628         2.89
 1.8           .587787         3.24
 1.9           .641854         3.61
 2.            .693147         4.

TIME:   0 SECS.
```

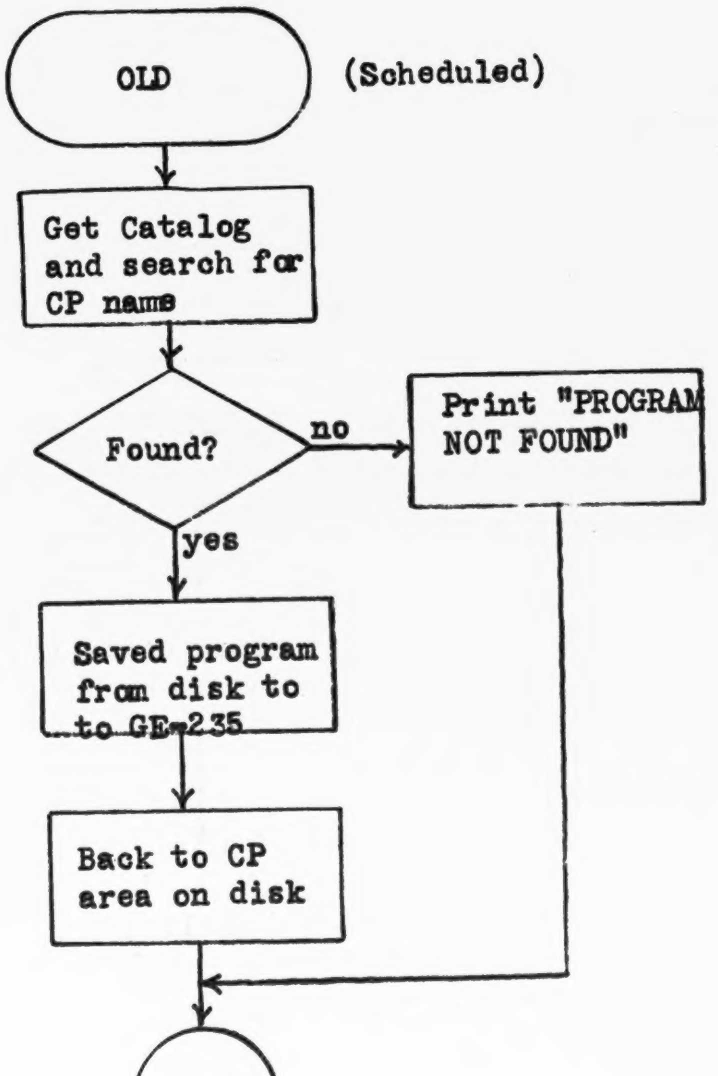
A partial list of system commands, and their functions, follows:

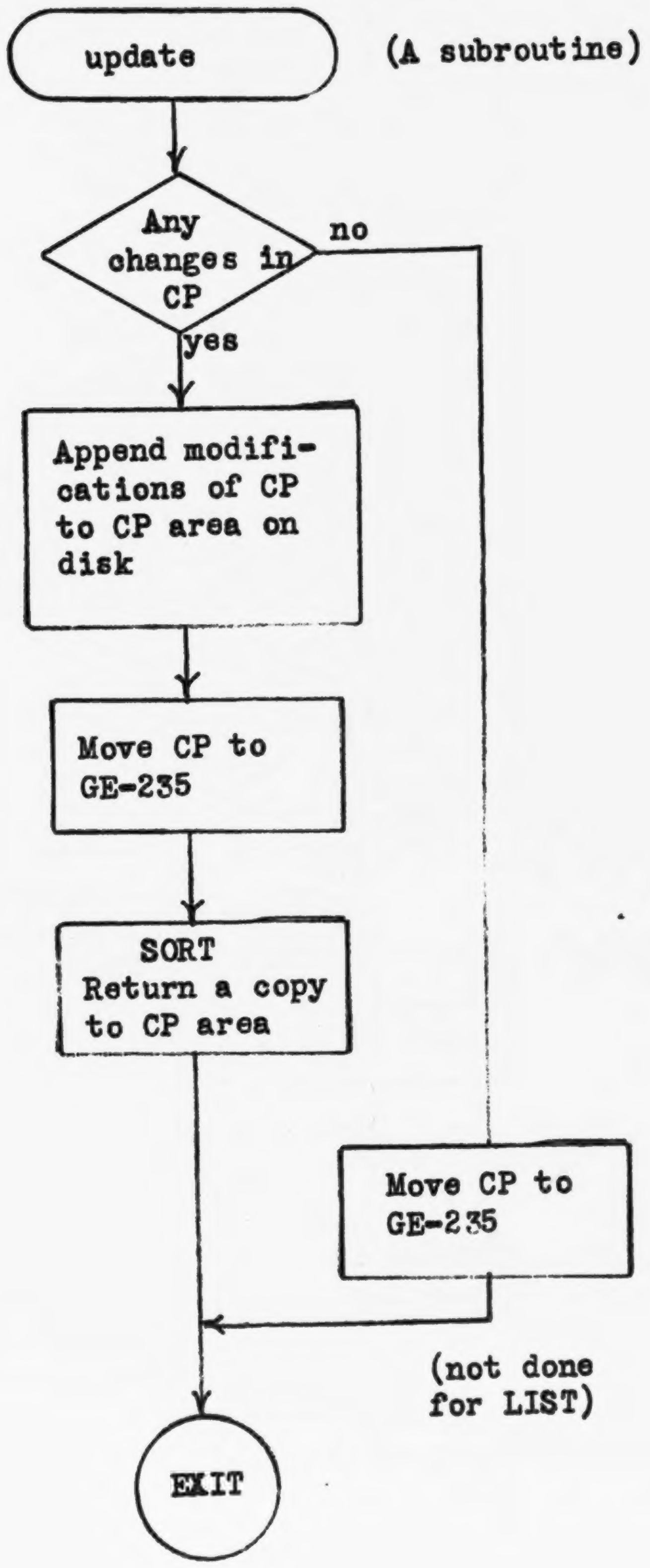
HELLO	The request for gaining access. The user must then supply his user number.
NEW	Clears out the "current program" giving the user a clean slate for composing his new program.
RUN	Run the current program, using BASIC unless some other language has been specified. This causes both compilation and execution.
LIST	The current program is printed out on the teletype.
SAVE	The current program is saved, and a catalog entry is made under the user's number. The current program is not destroyed.
UNSAVE	The catalog entry for the current program is removed.
OLD	A previously saved program is made into the current program. The saved program is not changed.
CATALOG	A listing of all program names in the user's catalog is printed on the teletype.
RENAME	The user is allowed to rename a program without destroying it.
GOODBYE	Signs the user off the computer.

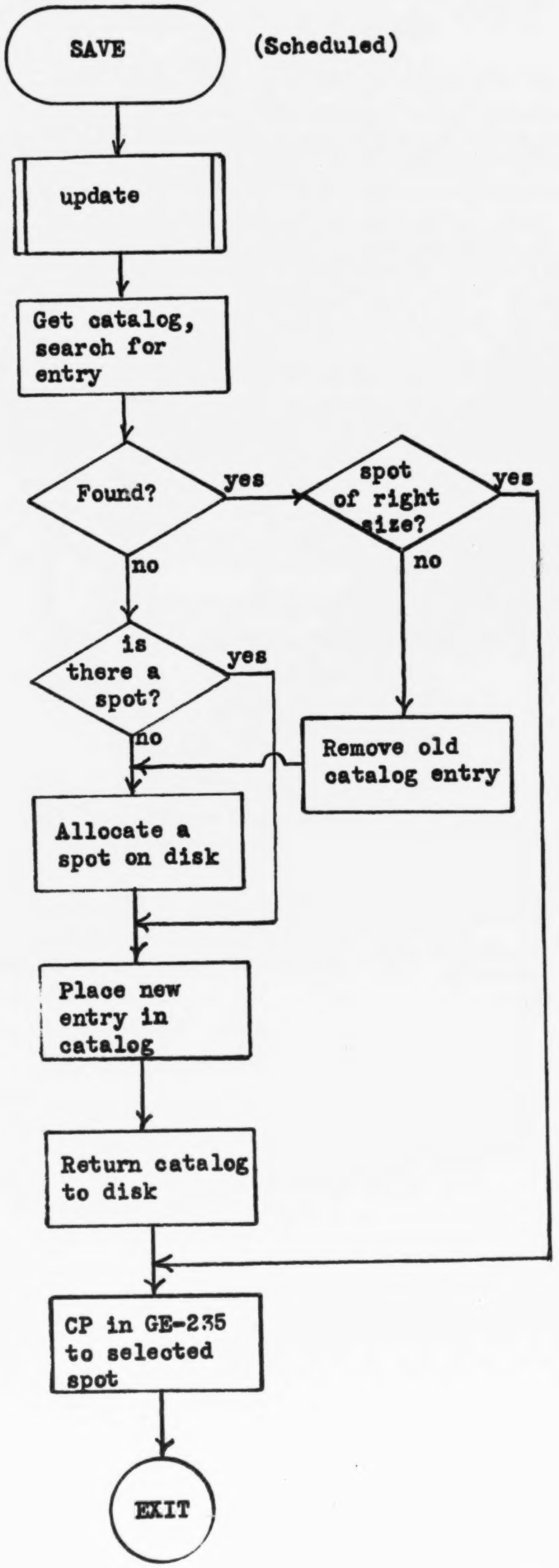
There are other commands, but the majority of users use no more than these most of the time. Block diagrams of some of the commands are included.

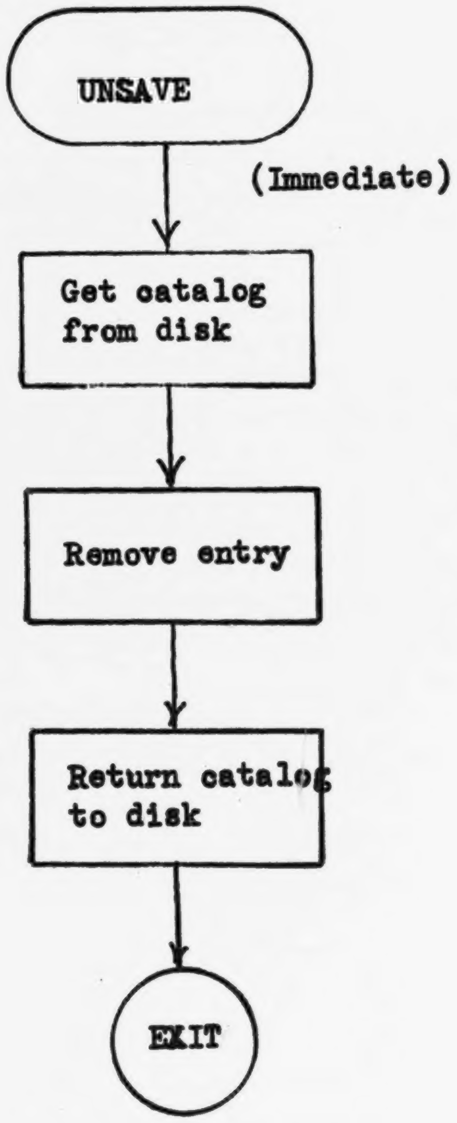
There is a library provision in the system. If, when retrieving an OLD program, the userappends three asterisk's to the name of the program, the search for it will be made in the catalog of the public library rather than his own private catalog. Once he obtains the program, he can treat it just as if he himself had just composed it. Of course, he cannot SAVE it back into the public library; if he types SAVE, it will be saved in his private library.

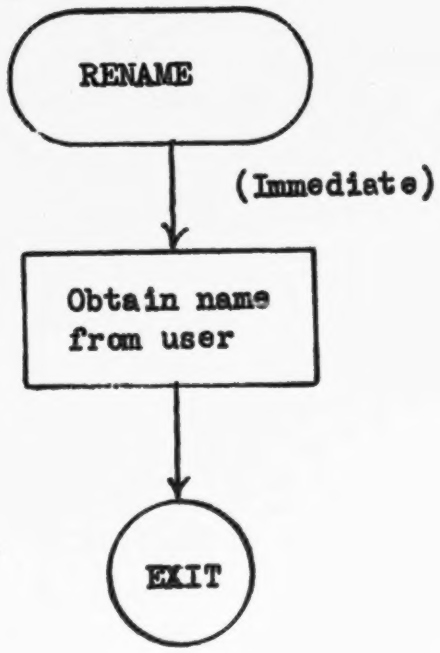

NEW
(Immediate)
Clear out Current Program (CP)
Obtain Name from user
EXIT

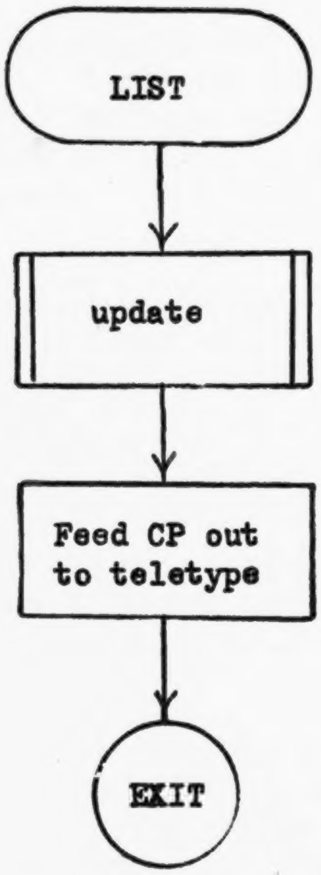

OLD
(Scheduled)
Get Catalog and search for CP name
Found?
no
Print "PROGRAM NOT FOUND"
yes
Saved program from disk to to GE-235
Back to CP area on disk
EXIT


update
(A subroutine)
Any changes in CP
no
yes
Append modifications of CP to CP area on disk
Move CP to GE-235
SORT Return a copy to CP area
Move CP to GE-235
(not done for LIST)
EXIT

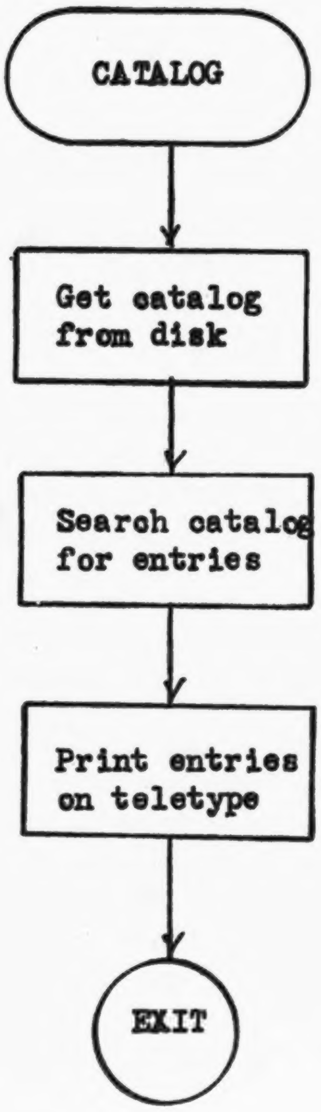

SAVE
(Scheduled)
update
Get catalog, search for entry
Found?
yes
no
spot of right size?
yes
no
is there a spot?
yes
no
Remove old catalog entry
Allocate a spot on disk
Place new entry in catalog
Return catalog to disk
CP in GE-235 to selected spot
EXIT


UNSAVE
(Immediate)
Get catalog from disk
Remove entry
Return catalog to disk
EXIT

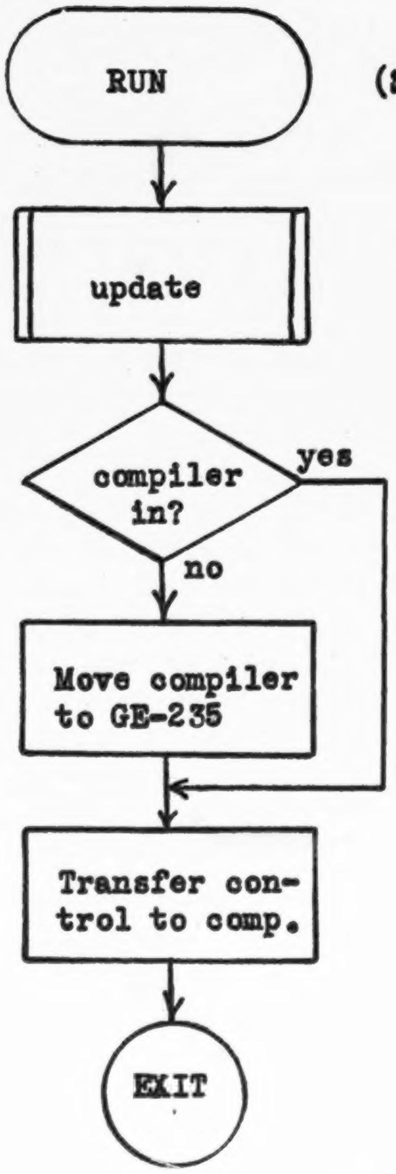

RENAME
(Immediate)
Obtain name from user
EXIT


LIST
update
Feed CP out to teletype
EXIT

(Immediate unless update is needed, then scheduled.)


CATALOG
Get catalog from disk
Search catalog for entries
Print entries on teletype
EXIT

(Immediate.)


RUN
update
compiler in?
yes
no
Move compiler to GE-235
Transfer control to comp.
EXIT

(Scheduled.)

In addition to the language BASIC, which is described in Appendix II, the system permits other languages to be used. An ALGOL compiler was prepared and is available in the system. An interpretive machine language called TSAP is also included and permits both training of machine language programmers and small portions of machine language programming itself.

A very important capability is provided by the Edit system. It is used by typing the command EDIT followed by the name of the particular edit function desired. For instance, the line numbers in a program may be resequenced by typing

```
EDIT RESEQUENCE     100, 0, 5
```

The numerical parameters in this case are to be interpreted as follows: "To the first line in the current program whose line number is not less than 0, assign line number 100, and proceed from there in steps of 5."

Two programs may be merged using the MERGE or WEAVE functions. These functions are useful, for instance, when a student wishes to merge into his program a set of "official" data provided by the instructor. EXTRACT and DELETE retain or discard, respectively, portions of the current program according to line number ranges specified. Listing either forward or backward of specific portions of a program may be obtained using the LIST function. There are other functions, but a detailed discussion is not included here.

For certain jobs a teletype machine may not be adequate. The job may require magnetic tapes, or need to read cards or print large amounts of output. A "background" mode is provided in the system. It is controlled from the console typewriter, and permits the use of all peripherals. Background programs share time with time-sharing, but on a lower priority. It is thus possible to run jobs that require peripherals concurrently with time-sharing. Such jobs might include, for instance, assemblies of compilers or certain administrative tasks such as a disc-utilization survey.

It should be noted that almost all of the software for the system was prepared by Dartmouth undergraduate students at Dartmouth College under the general supervision of the Director and Associate Director of this project. Included were the executive systems for both computers, the several compilers, the editing system, and the background capability. Even the machine language assembly program was extensively rewritten to cut assembly times by more than 75 percent.

The New System

Currently under development, a new Dartmouth Time-Sharing system will provide for many more (up to 200) simultaneous users. It is being built around a GE-625 computer system. The hardware configuration and the structure of the executive system is different from the original system, but the picture to the ordinary user will almost exactly be the same.

The hardware configuration is shown in figure I-B. Several important differences exist between the two systems. With the new hardware there is an adequate method of memory protection and hardware relocation of program. Thus, it is planned to retain in the core memory the programs of several of the current users. Those not in core will be temporarily retained on a magnetic drum, which can be accessed in a much shorter time than can a disc. Another difference is that the Datanet-30's in the new system are concerned only with message switching between the main compute system and the teletypes. They will also take care of character and line deletes. But the main executive functions of scheduling jobs, controlling the composing of a new program, carrying out the commands, and managing the use of the system reside in the same computer as do the user programs. (However, it is not convenient at this time to supply memory protection for compilers, which must reside within the same protected area as its work. Therefore, until this condition is remedied, the compilers will not be reentrant even though users will be fully multi-programmed.)

The user will use the same commands as before. But they will be handled differently. The reason is that the new system will be more general and will be capable of expansion as new ideas come along.

All teletype lines pass through the Datanet-30. All lines which are not in actual operation with BASIC or some other system will be logically connected to the LOGIN module of the executive. Thus, a user can type HELLO, go through any validation procedures required by LOGIN, and request a certain system. The LOGIN module then disconnects itself from the "line" and connects it to the Simple Monitor, which analyzes all subsequent commands such as OLD, NEW, and LIST. In other words, the SM portion of the executive system plays the role played by the command and control portion of the Datanet-30 program in the original system. When RUN is typed, control is passed to BASIC. After the run is completed, control reverts to the Simple Monitor. If GOODBYE is typed, the user's work is cleared out, and his teletype line is logically returned to the LOGIN module to wait for another HELLO.

It is planned to include in the new system all the major services currently provided by the "old" system. In particular, the interface provided the ordinary user will remain simple. Additional features will be added to enhance the research potential of the new system, and these will not erode the simple and friendly interface whose value has been so resoundly proven by this project.

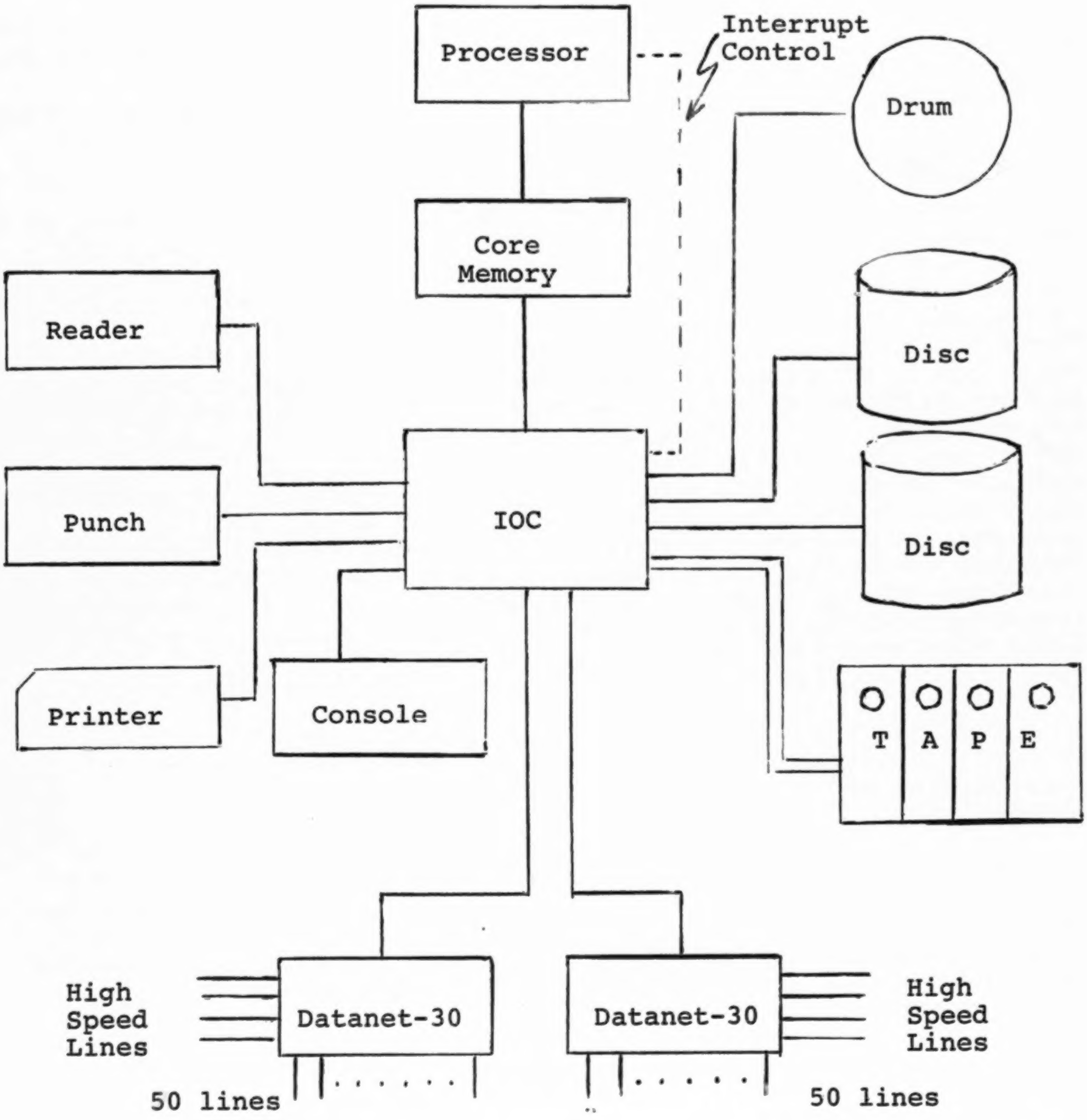

Processor
Interrupt Control
Drum
Core Memory
Reader
Disc
Disc
IOC
Punch
Printer
Console
T A P E
High Speed Lines
Datanet-30
Datanet-30
High Speed Lines
50 lines
50 lines

Figure I-B. The hardware configuration of the GE-635 system that is to replace the GE-265 system.

APPENDIX II

The BASIC Language

1. Introduction

The single most important factor contributing to the success of this project is the simple language BASIC. It is very easy to learn, and can be taught or self-taught in several hours or less to not only college freshmen but also high school students. Faculty members having no previous computing experience have been able to learn it, as have businessmen and administrators. A number of fourth and fifth graders have taught themselves this language. In short the BASIC Language has removed "programming" from the private domain of the experts, and, for much less effort than learning to type or drive a car, has placed the powers of the computer into the hands of students and scholars. For the first time, large numbers of users are now using computing to enhance and accelerate their education and research.

2. General Description

BASIC is an algebraic language with a strong external resemblance to other algebraic languages such as FORTRAN and ALGOL. It differs, however, in certain seemingly unimportant ways, and these differences are what has contributed to its simplification and resulting success. In some cases these differences represent restrictions over other algebraic languages, but it is our experience that most of the time these restrictions do not limit the applications a user can make with BASIC. In other cases the differences in BASIC reflect improvements on and generalizations of other languages, particularly where other languages were found to be unduly and artificially restrictive. Some of these differences are discussed in this section.

BASIC is a line oriented language with each line beginning with a line number. This line number serves not only as a serial number for editing purposes but also as a statement label. In a time-sharing environment, line numbers for editing purposes are essential; therefore, in BASIC we allow them to serve also as statement numbers thus avoiding the necessity for teaching a separate statement labeling technique. The restriction here is that symbolic statement labels are not allowed, but neither are they allowed in FORTRAN.

Perhaps the single most important feature of BASIC is the automatic or implied declaration of all variables, both simple and subscripted. Simple variables are declared by their appearance in the program. There is no distinction between real or integer as in both

FORTRAN and ALGOL. Subscripted variables, both singly and doubly, can also be declared by implication through their appearance in the program. The convention is that a singly subscripted quantity upon first appearance is assumed to be a vector having subscript range 0 through 10. A doubly subscripted quantity is assumed to represent a matrix with both subscripts from 0 through 10. If different maximum dimensions are required for a specific problem, the programmer may use a DIM or dimension statement. However, most programs turn out to require only small vectors and matrices and therefore no special dimension statement.

The programmer is restricted to simple variable names consisting of a letter or a letter followed by a digit. Subscripted quantities must be named by a single letter. The same letter may represent both a simple variable and a subscripted quantity, though not both a singly and a doubly subscripted quantity. BASIC ignores all spaces so that the programmer need not be concerned about such formatting as is required, for example, by many card-oriented systems. The representation of all numerical constants is free format. As stated earlier, no distinction is made between integer constants and more general constants. Therefore the mixed expression problem of most early versions of FORTRAN does not exist in BASIC.

At many places in the definition of the BASIC Language and the definition of the interpretation of BASIC operations, an attempt is made to provide the interpretation that the beginning user was likely to expect rather than the one that the sophisticated programmer might demand. Thus, for example, in raising a number or quantity to a power, if the power happens to be an integer the appropriate number of multiplications is performed. If the power happens to be a non-integer, the result is determined by taking logarithms and exponentials. Thus, negative quantities raised to odd powers will remain negative, just the way the ordinary user would expect.

All BASIC statements consist of a line number followed by a word which identifies the type of the statement; the statement ends with the end of the line. This simple structure of BASIC easily permits very fast single pass compilers. It is thus possible to retain for the user his program in its source language BASIC. Each time the user calls for a RUN, he automatically receives an edit of his program followed by a compilation followed by an execution. The compilation takes place in a time roughly equal to the editing time.

Several features included in the Dartmouth Time-Sharing System permit an exceptionally easy use of the BASIC Language and are often identified by users as being part of BASIC. These are the ability to erase a character or delete a line with a single stroke on the keyboard of the teletype machine. Another feature is the ability to change a line in the middle of the program simply by retyping the line in question. Lines can be deleted by typing the line number without anything following it. Lines can be inserted by typing them with a line number that lies in between two line numbers in the program. The BASIC program is thus retained inside the computer system, and the user constructs it and edits in a very simple way from the teletype keyboard.

3. Elementary BASIC

We have found it convenient in teaching BASIC to identify as elementary BASIC nine BASIC statements. With these nine statements most programs can be written quite conveniently. These are not presented as minimal set--the looping statements are included because looping is an extremely common programming technique. The input-output statements in elementary BASIC are READ, PRINT, and DATA. The READ statement causes the variable following it to be assigned data according to the next consequtively available data in the data block. The data block is prepared by collecting together all the numerical data appearing in the DATA statements, independently of in which data statement a particular number is found. The PRINT statement causes the values of the variables or expressions following it to be printed. In both read and print statements the comma is used to separate the items. Printing is done in a pre-determined, six significant figure format. The type of the format is determined not by the user but by the range and the type of the number involved. That is, integers come out as integers. Non-integers come out as decimal numbers with the decimal point appropriately placed. Numbers out of range are represented as a fraction and an exponent. The print statement automatically places the printed numbers in columns fifteen spaces wide. On a teletype machine there are five fields. If the user wishes to pack information more closely on a line, he can use the semi-colon rather than a comma to separate the items in his list. The user may also print verbal information by enclosing the information inside quotation marks in a print statement. Thus, labeling and numerical information can be mixed. A print statement without any arguments following it will print a blank line.

The Computational or assignment statement is introduced with the LET. Following the word LET is a variable, which can be either simple or subscripted, and then an equal sign. To the right of the equal sign is an arithmetic expression which is to be evaluated. The expression may be quite complex, and the meaning of the parenthesis and the order of precedence for the arithmetic operators follows standard usage. Thus, multiplications are performed before additions unless parenthesis intervene. The multiply symbol is an asterisk, and simple juxtaposition of variables will *not* cause them to be interpreted as a product, but will cause an error. The exponentiation or raise-to-a-power symbol is the up-arrow. On an ordinary teletype machine superscripts and subscripts as such are not easily obtainable.

The BASIC statements are normally executed in sequence. A GØ TØ statement can be used to transfer control to the statement whose number follows the GØ TØ. The conditional transfer is carried out by the IF statement which has the following form:

IF <relational expression> THEN <line number>

If the relational expression is true, then control is passed to the line number indicated after the THEN. If the relational expression is false, then the computer continues on its sequence. All six common relational operators are included. Less-than-or-equal is represented by a less-than sign followed by an equal sign, and similarly for greater-than-or-equal. Not-equal is represented by a less-than sign followed by a greater-than sign.

Looping is carried out by a pair of statements: the FØR statement and the corresponding NEXT statement. The NEXT statement serves to define the scope of the loop. In this respect it is similar to the CØNTINUE statement in FORTRAN. The FØR statement indicates a variable which it controls together with an initial value, a final value, and a step size. If the step size is not specified, it is taken to be unity. The corresponding NEXT statement must refer to the same control variable. The test for completion of the loop is made at the beginning of the loop. Thus, if the loop is in fact vacuous, control then jumps around the body of the loop to the statement following the NEXT. The initial, final, and step size values may be anything. Thus a loop can operate with a decreasing series of values as well as an increasing series, or with negative numbers as well as positive numbers. The initial, final, and step size values are determined upon entry to the loop and thereafter remain unchanged. It is thus not possible to change the step size in the middle of the loop if one is interested in preparing a table with differing step sizes as the table progresses.

The final statement in the program is always an END statement.

Besides the nine statements, BASIC includes a number of standard functions, of which most are listed below:

SIN
CØS
TAN
ATN
SQR
RND (random number)
LØG
EXP
ABS
SGN (sign -1, 0, +1 of argument)
INT (integer part)

An example program utilizing all nine of the elementary BASIC statements is shown on Page 8 . It calculates the sum of a geometric series for 100 terms given the initial term and the ratio. (For a more complete description of the details of the BASIC Language, the reader is referred to the Manual for BASIC.) The program should be self-explanatory, and is designed primarily to illustrate the nine statements of elementary BASIC. The sample run is shown on page 8.

It is revealing also to compare a typical program in BASIC with the corresponding program in FORTRAN and ALGOL. The program chosen is a simple one that prints a small table of the square and cube roots of numbers from 1 to 2 with a spacing of .1. The programs in all three languages are shown on Page 9 .

The BASIC program should be self-explanatory, and has been actually test-run. The ALGOL shown is the version of ALGOL available on the Dartmouth Time-Sharing System, and has also been test-run. The FORTRAN program is in a hypothetical version of FORTRAN that might quite reasonably be implemented on a Time-Sharing System, but which does not, to our knowledge, actually exist. However, its form follows closely the form and conventions of standard FORTRAN II.

Comparing these programs, we see that BASIC is economical in typing, and is simple to read. The FORTRAN version contains several extra steps required by the peculiarities of FORTRAN. For instance, non-integer step sizes cannot be used in DO statements. The distinction between real and integer requires both extraneous decimal points and the extra statement in line 30. Finally, the FORMAT statement is awkward to use when only simple output is needed.

The ALGOL version shows an economy of typing similar to BASIC's, but additional declaration statements are needed. Furthermore, even for relatively simple looping statements, the compound statement structure using the BEGIN-END is needed.

This simple example thus shows that BASIC contains crucial simplifications over both FORTRAN and ALGOL. These simplifications, and others of a similar nature, have turned out to be the important difference in being able to bring computing to essentially all students and faculty at Dartmouth.

4. Advanced BASIC

The title of this section is misleading in that the statements outlined in Section 3 can lead to quite complex programs. There are additional statements, of course, and these are what is discussed here. Perhaps the title "less-needed BASIC statements" would be more appropriate.

There is an additional input instruction called INPUT. It operates much as READ does, but draws data from the teletype keyboard rather than from the DATA block. This statement permits the user to interact with his program while it is running. It is essential for game-playing, or any similar application. For easy-to-use utility programs, it is easier to have the lay user enter his special data through an INPUT statement rather than giving him directions for attaching DATA statements to the programs before running it.

Besides the special functions automatically provided by BASIC, the user may define up to 26 functions of his own through a DEF or define statement. The format is:

DEF FN <letter> (<simple variable>) = <expression> .

The name of the function is thus three letters of which the first two must be FN. The parameter in the DEF statement must be a simple variable, that is, a letter possibly followed by a digit. Defined functions are used exactly like ordinary functions.

Certain sections of a program may be transferred to using the GØSUB statement, which also "remembers" the return address. The RETURN statement in the sub-program causes a return to the statement following the GØSUB. GØSUB's may appear in sub-programs; that is, they may be nested.

The RESTØRE statement causes the pointer in the data block to be reset so that the next READ statement will read the <u>first</u> datum as if it were the first READ statement in the program.

The DIM statement is used to declare sizes for list and tables (vectors and matrices) other than the standard implied declaration (0 to 10.) It may be used to declare large matrices, or to conserve space in a tight program by declaring the matrices to be small.

The REM statement is used to supply remarks or comments in the program. Whatever follows REM is ignored until a carriage return is reached. The STØP statement is equivalent to a GØ TØ to the line number of the END statement in the program.

Besides these additional instructions, the PRINT statement is capable of more generality. Using the ; instead of the , causes a "packing" in the output line. For labels, the packing is simple juxtaposition. For numbers, the packing depends on the number of digits or special characters printed out. This should not be confused with formatted output, which can be provided by special GØSUB type routines if needed.

5. Matrix Operations

One of the most useful of all BASIC statements is the MAT. It indicates that what follows is to be interpreted as an operation on a matrix (or vector.) With MAT statements, matrices can be added, subtracted, or multiplied, scalar multiplication can be performed, or an inverse or a transpose produced each in one line. Also an identity matrix or a matrix of all zeros or ones can be supplied, or a matrix read or printed. For instance, the following program exclusive of DIM and DATA statements can find the solution of a set of linear equations:

```
10  MAT READ A,B
20  MAT C = INV(A)
30  MAT D = C*B
40  MAT PRINT D
```

Details of MAT operations can be found in the BASIC Manual.

6. Future Plans

As the use of BASIC has increased, we have become aware of a number of shortcomings in the language. These range from a poor original choice of convention to the inability to handle alphabetical information as data. Accordingly, as BASIC is being implemented for the new 635 computer system, a number of changes are being made.

a. A new statement ØN

A new statement ØN will act as a many-way branch. For example:

```
117 ØN X + 2 GØ TØ  100, 200, 300, 450
```

will transfer to 100 if X + 2 (truncated) equals 1, to 200 if X + 2 (truncated) equals 2, and so on.

b. Defined functions can have any number of arguments.

Functions defined by the DEF statement will be able to have any number of arguments, including none. Two argument functions will, for instance, greatly simplify the DIFFEQ student exercise described in Appendix III.

c. A new statement RANDØMIZE

This statement will jumble up the random number sequence. After a program is debugged (for which it is important to have a reproducible sequence), the user can insert a RANDØMIZE statement to avoid repeating earlier experiments or similar experiments by other users.

d. String-handling capability

The new BASIC will have the ability to read, input, print, assign, and compare string data. Comparison will be on the basis of the lexeographical order implied by the ASCII code. Individual characters in the string can be retrieved or modified using a special CHANGE statement, which spreads out a string into a numerical vector, one character per entry, or vice versa.

e. MAT clean-up

Certain conventions in the original MAT package proved to be unwise. In particular, subscripts ran from 0 to N. In the new BASIC, subscripts will run from 1 to N. Further changes include implied dimensioning as well as implied declaration whenever possible. A programmer will be able to use the MAT instructions as easily as he can now use ordinary BASIC.

f. Miscellaneous changes

Certain small changes are being made in the use of the ; in PRINT statements.

```
LIST


SERIES    14:24   6/13/67 TUESDAY

100  PRINT "A", "R", "SUM", "TRUE SUM"
110  READ A, R
120  IF ABS(R) >= 1 THEN 210
130  LET S = 0
140  LET T = A
150  FOR I = 1 TO 100
160     LET S = S + T
170     LET T = T*R
180  NEXT I
190  PRINT A, R, S, A/(1-R)
200  GO TO 110
210  PRINT A, R, "SERIES IS DIVERGENT"
220  GO TO 110
230  DATA  1, .2, 1, .5, 1, .8, 1, .9, 1, 1.5
240  END

RUN


SERIES    14:25   6/13/67 TUESDAY

A              R              SUM            TRUE SUM
 1              .2             1.25           1.25
 1              .5             2.             2
 1              .8             5.             5.
 1              .9             9.99973        10.
 1              1.5           SERIES IS DIVERGENT

OUT OF DATA IN  110

TIME:   1 SECS.
```

X-BAS

```
10  PRINT "X", "SQUARE ROOT", "CUBE ROOT"
20  FOR X = 1 TO 2 STEP .1
30      LET S = SQR(X)
40      LET C = X↑(1/3)
50      PRINT X, S, C
60  NEXT X
70  END
```

X-FOR

```
10     PRINT 20
20     DO 10 I = 0, 10
30     XI = I
40     X = 1.0 + XI/10.0
50     S = SQRTF(X)
60     C = X**(1/3)
70 10 PRINT 30, X, S, C
80 20 FORMAT(1HX, 14X, 11HSQUARE ROOT, 4X, 9HCUBE ROOT)
90 30 FORMAT( 3F15.5 )
99     END
```

X-ALG

```
10  BEGIN REAL X, S, C;
20  PRINT("X", "SQUARE ROOT", "CUBE ROOT");
30  FOR X := 1 STEP .1 UNTIL 2 DO
40      BEGIN S := SQRT(X); C := X↑(1/3);
50      PRINT( X, S, C ) END END
```

Appendix III

The Freshman Training Program

The TEACH System

1. Problems from Freshman Mathematics

The computer training program at Dartmouth is given to students enrolled in their second course of freshman mathematics. There are two main options, depending on the student's interest, for this second course:

First Course — *Introduction to Calculus*
Second Course — *Techniques of Calculus* / *Finite Mathematics*

The Calculus option is elected by students interested in Mathematics, Engineering, Physics, and other Sciences. The Finite Mathematics option is elected by Pre-Meds and students headed toward the Social Sciences. (In addition, there are advanced placement tracts and honors sections, and the computer training is also provided for these modifications.)

The content of these three courses may be summarized as follows:

Introduction to Calculus

Differentiation
Curve Plotting
Integration
Simple Differential Equations

Techniques of Calculus

Integration Techniques (linear)
Techniques for Solution of Differential Equations
Infinite and Power Series
Applications

Finite Mathematics

Logic and Sets
Probability Up To Central Limit Theorem For Binomial Trials
Vectors and Matrices
Applications to Markov Chains or Linear Programming

An important slant of the training program was to present programming techniques in terms of applications to the mathematics course. We thus avoided the artificiality of teaching programming as an isolated discipline. In line with this point of view, we presented two extra one-hour lectures near the beginning of the course. The student was shown a teletype machine, and then shown how to use BASIC to carry out certain simple computations. At the end of the second lecture, he was handed his first assignment. Later on in the course he received on schedule his second, third and fourth assignments. While the first one is more of an ice-breaker and is not directly related to the course, the remaining exercises are related to the course both in content and in timing.

For the Calculus option, the four computer exercises are:

PIE	The student is asked to approximate π by calculating the perimeter of a regular polygon inscribed in a unit circle. He starts with a hexagon and doubles the number of sides a specified number of times. The most common type of program for this problem uses the BASIC statements READ, DATA, PRINT, LET, FOR, NEXT, and END.
TRAP	The student is asked to prepare a program for implementing the trapezoid rule for approximately a definite integral. The end points of the interval as well as the number of subintervals are supplied as data, and the function is introduced with a DEF statement. The usual approach will include a FOR statement with a non-integer step size, though other techniques for running through the sum are acceptable.
SINE	The student is asked to construct a program to explore the truncated power series for the sine function.
DIFFEQ	A program is required for integrating a first order differential equation with a given initial value. The method used is the so-called "modified Euler" which also can be viewed as a simple type of Runge-Kutta method. In any case, this method is stable, and has second-order error term for a fixed range of integration. The method is thus "safe" for general use, and halving the meshsize reduces the error to approximately one-quarter.

For the finite mathematics option, the four problems are:

MØD	The student becomes familiar with modular or residual arithmetic, and is asked to calculate

	the value of AB mod (M). He will probably subtract multiples of M until the residual is < M; he has not yet seen the INT function, which is the natural way for performing this calculation. He will probably use the BASIC statements READ, DATA, PRINT, LET, IF and END, and possibly also the FOR and NEXT.
QUINT	The student must write a program for finding a real root of a given quintic equation, which is introduced through a DEF statement. The method requested is a binary search between 0 and 1 using N (read in) iterations.
BDAY	The student must program the formula for the probability that two or more in a group of N persons have a birthday on the same date.
OZ	In this exercise the student must simulate a three state Markov chain purportedly dealing with weather in the Land of Oz. He must use the RND function for generating random numbers between 0 and 1, and will probably elect to retain the transition probabilities in table (matrix) form. He is asked to accumulate the elementary statistics from the simulation.

Slightly edited instruction sheets for each of the eight exercises are included. These follow a one-page general instruction sheet handed to students at the end of their second computer lecture.

The general instruction sheet is shown on page 4 of this Appendix, and the instruction sheets for the eight exercises are included on pages five through eleven.

MACHINE TESTING OF COMPUTER PROBLEMS

The Dartmouth College Computation Center has developed a special system for aiding freshmen in their required programming problems. Each of the four assigned problems can be automatically tested by the computer, to help students debug their programs. To enable the computer to do this, a small number of conventions must be observed.

Name of problem: Each problem has an official name. The student must use precisely this name for his program.

Data: The instructions for the problem will specify what data is to be read. The program must read this data, and no more. For example, if two numbers M and N are to be read, any attempt to read only one number, or read more than two, results in an error message. If the programmer wishes to try out his program for various values of M and N, he can do this by several RUN's, each with different data.

Answers: The instructions will specify what answers are to be computed, and what names to call them. Using names other than the official ones will result in an error message. NOTE: The machine will test the computed answers, not the printed ones. Thus it is possible to obtain an error message even if the answers printed are correct.

Printing: Answers may be printed in any format convenient to the programmer (see note above). To avoid confusing printed answers with computed answers, it is recommended that all answers be printed after the computations have been completed.

Line Numbers: No line number greater than 9999 is allowed.

How to test: Write the program and debug it in the usual manner. When you think that it works correctly, type the word TEST. The machine will then either approve your program, or it will give a hint as to where you have made a mistake.

Hand In: Always hand in a LIST, with your name and course number, a sample RUN, and TEST showing approval of your program.

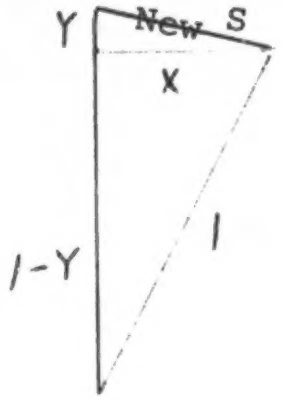
FRESHMAN MATHEMATICS

Computer Problem No. 1.

Name of problem: PIE

Purpose: To obtain an approximate value for π, by approximating the circumferance of a unit circle.

Mathematics involved: Start with a regular hexagon inscribed in the unit circle (6 sides, each of unit length). If we know the number of sides (N) and the length of the side (S), we obtain the same quantities for a regular polygon with twice as many sides as follows:



$X = S/2$

$Y = 1 - \sqrt{1 - X^2}$

New $S = \sqrt{X^2 + Y^2}$

New $N = 2N$

$P = NS/2$

where P is the approximate value of π.

Data: Read a single number D. This tells you how many times to double the number of sides. For example, if D = 2, you start with a hexagon, then double to 12 sides, and finally double to 24 sides.

Answers: Compute N (number of sides), S (length of side), and P (approximate value of π). Use the method described above; certain short-cuts can lead to bad round-off errors.

Hand in: A LIST of your program (with your name and course), a RUN (using D = 10), and a TEST which approves your program.

FRESHMAN MATHEMATICS

Computer Problem No. 2.

Name of Problem: TRAP

Purpose: To evaluate the integral

$$I = \int_A^B f(x)dx$$

by the trapazoid rule. The mathematics of this will be covered in class.

Data: Read three numbers A, B, N. The first two specify the endpoints of the interval [A,B]. You are to divide the interval into N equal parts for the trapezoid rule.

Answer: The answer is I, the approximate value of the integral.

The function: To write a program that will work for an arbitrary function, one uses the DEF instruction. This works as follows: If at the beginning of your program you write

```
10   DEF   FNF(X) = EXP(X↑2)
```

then you may use FNF as if it were one of the usual functions in BASIC. Each time in your program that you have FNF(X), it will evaluate it as e^{x^2}. And of course, you may write FNF(A), or FNF(2.5), etc.

You should introduce your function as indicated. Then, in the rest of the program, always use FNF in place of the function. This will mean that you can apply your program to any other function by simply changing the DEF.

Hand in: As usual, hand in a LIST, a RUN, and an ok TEST.

For the run you are to compute

$$\int_0^1 e^{x^2} dx$$

to 6 place accuracy. You will have to try various values of N, till you get no further improvement. Note that this integral cannot be evaluated by a formula!

✓

FRESHMAN MATHEMATICS

Computer Problem No. 3.

Name of the problem: SINE

Purpose: To approximate sin (X) by taking M terms of the Taylor series (around X = 0).

Data: Read two numbers, X and M. X is the point at which we wish to approximate sin(X). M is the number of terms to be used in the Taylor series. For example, if M = 2, we approximate by $X - X^3/3!$

Answers: Two numbers are to be computed. T = the Mth term of the Taylor series. S = the sum of the first M terms -- hence our approximation.

Hand in: A LIST and an ok by means of TEST.

In addition we suggest that you RUN a few sample approximations. Get a feeling of how many terms you need in the series if X is small, and how many if X is around 3, for an approximation to 4 significant figures. You may also want to see whether $\sin(X) = \sin(X+2\pi)$ can be verified from the series.

FRESHMAN MATHEMATICS

Computer Problem No. 4

Name of the problem: DIFFEQ

Purpose: To find that solution of the differential equation

$$y' = f(x,y) \tag{1}$$

which passes through the point (a,y_0). Or, more precisely, to find the value of the unknown function y(x) at the point x =b.

Mathematical method: Divide the interval [a,b] into n equal parts, each of length h. Let x_k be the kth end point, i.e., $x_k = a+kh$, for $k = 0, 1, \ldots, n$. And let $y_k = y(x_k)$. At the start $x = x_0 = a$, and $y = y_0$. Given an approximate value for y_k, we compute y_{k+1} in two steps:

$$p_{k+1} = y_k + hf(x_k,y_k) \tag{2}$$

$$y_{k+1} = \frac{1}{2}(y_k + p_{k+1} + hf(x_{k+1},p_{k+1})). \tag{3}$$

Then $y(b) = y_n$. The derivation of these formulas is given in the enclosed pages.

Data: READ the numbers A, B, YO, and N. They play the roles of a, b, y_0, and n above.

Conventions: Use P and Y for the current values of p_k and y_k at each iteration. Since BASIC does not have defined functions with two variables, one introduces f(x,y) by a trick:

DEF FNF(Y) = ...

with the formula for f(x,y) on the right. As long as X has the desired value before FNF is used, one need only type FNF(...) where ... contains the desired value of Y. A newer version of BASIC contains provision for defined functions of several arguments.

Answers: At the conclusion of your computations PRINT N and Y. The latter should at this stage be y(b).

Hand in: A LIST and an ok by means of TEST. Also hand in a RUN in which FNF(Y) = Y+EXP(-X), A=0, B=1, YO=1, and N=100.

Note: This is a very useful program. You may wish to try it out on various differential equations, and may want to print out the values of the function y over a whole interval.

MATHEMATICS 6

Computer Problem No. 1

Name of problem: M0D

Purpose: To obtain the product AB (Mod M), that is the product of A and B, if we want only the remainder after dividing by M.

Data: Read three numbers, A, B, and M, in that order.

Answer: Compute P, where

$$P = AB \ (\mathrm{Mod}\ M).$$

Mathematics involved: Modular arithmetic, or arithmetic modulo a given number M, is integer arithmetic in which multiples of M are always discarded. Thus the only numbers are 0, 1, 2, ..., M-1. Otherwise one can operate in modular arithmetic very much the way one operates in ordinary arithmetic. Thus the present problem one computes AB, and then reduces the answer (if necessary) by discarding multiples of M.

Hand in: A LIST of your program, a RUN, and a TEST which approves your program--all on the same piece of paper.

MATHEMATICS 6

Computer Problem No. 2

Name of problem: QUINT

Purpose: To find a root of the quintic equation

$$x^5 + 2x^3 - 1 = 0.$$

Data: Read a single number N, the number of iterations.

Defined function: Introduce the quintic as a function, by an instruction of the form

```
DEF FNF(X) = ...
```

Once you have done this, you may use FNF(X) anywhere in your program to give you a value of the quintic.

Mathematics involved: We note that the quintic has a negative value at $x = 0$, and a positive value at $x = 1$. Let A = 0, and B = 1. There must be a root of the equation between A and B. Let X be half-way between and evaluate the quintic at X. If $FNF(X) < 0$, then we have a root between X and B. If $FNF(X) > 0$, then there is a root between A and X. In either case the interval has been cut in half. We again choose X as the midpoint, and start the second "iteration", which proceeds just as before. If we do this 20 times, the interval has been cut to about .000001, and hence we know a root of the equation to within this accuracy!

Answer: At each stage A = left-hand end-point, B = right-hand end-point, and X = mid-point. The final value of X is the answer. Your program should carry out precisely N iterations.

Hand in: A LIST (with your name), a RUN, and a TEST--all on the same piece of paper.

MATHEMATICS 6

Computer Problem No. 3

Name of problem: BDAY

Purpose: To find the probability of two out of N people having the same birthday.

Data: N = number of people.

Answer: P = probability of two people having same birthday.

Mathematics involved: See INTRODUCTION TO FINITE MATHEMATICS, Chapter IV, Section 4.

Hand in: A LIST, a TEST, and four RUNs, using

N = 22, 23, 30, 50.

Compare your answers to the answers in the text.

MATHEMATICS 6

Computer Problem No. 4

Name of problem: 0Z

Purpose: To simulate a Markov chain -- The Land of Oz.

Data: N = number of days.

Procedure: Start with RAIN. Then simulate N days, keeping a count of the number of days of each type. The transition matrix is:

$$\begin{array}{c|ccc} & \text{RAIN} & \text{NICE} & \text{SNOW} \\ \hline \text{RAIN} & .5 & .25 & .25 \\ \text{NICE} & .5 & 0 & .5 \\ \text{SNOW} & .25 & .25 & .5 \end{array}$$

Random numbers: You may generate a random number by:

LET A = RND (X).

Then A will be less than .3 with probability .3, it will be between .3 and .5 with probability .2, etc.

Convention: Always let a low value of RND correspond to RAIN, the middle range to NICE, and large values to SNOW.

Answers: The number of times in various states -- not counting the initial state (RAIN.)
The fraction of time in each state.
Compare these fractions with the long-range probabilities: .4, .2, .4.

TEST: There is no TEST program available for this problem. However, If you RUN your program for N = 30, you may compare it with your homework problem.

Hand in: A LIST, and a RUN for N = 2500.

2. The TEACH System

Upperclass student readers are used to check the regular homework exercises of the students. However, this method for checking computer exercises is not practical. First of all, both the method and the answer must be correct--there is no partial credit. Secondly, there will be many possible correct programs, too many to ask any one person to pass judgement on. It was thus necessary to develop a machine method for testing student program.

The TEACH System is actually an extension of BASIC. It operates by appending to the student program a checking program, making changes to the student program, and then executing the combination. In effect, the TEACH test program is "running" the student program with special test data. If the correct answer is obtained in all cases tried, the student program is assumed to be correct. If one or more cases produces an incorrect answer, the test program will print out an error message that can be made dependent on the particular incorrect answer produced. Thus, the TEACH test program not only accepts a correct program, it also assists the student by supplying error messages that suggest what is wrong with the student program. The matter of identifying the nature of the error in the student program from the particular wrong answer produced is limited only by the ingenuity of the instructor in anticipating the kinds of mistakes that are likely to be made.

Specifically, to use the TEACH System, the student must:

1. Prepare his program according to the instructions, using specified variables for his "answers."
2. Use line numbers of 5 digits or less.
3. Have his program treat one case at a time, with no doubling back for multiple cases.

He composes and debugs his program in the usual way. When he feels that it works, he types TEST to invoke the TEACH system. The following steps take place:

1. A TEACH program having the same name as his program is appended to the student's program after the END statement.
2. All PRINT, DATA, STØP and END statements are removed from the student's program. STØP and END statements are replaced by GØTØ's to the first instruction in the TEACH section.
3. The combined program is started running at the beginning of the TEACH section.

Several conventions were adopted to make it possible to carry out a very flexible check of the student's program.

1. The symbol * stands for the first executable instruction in the student's program. Thus the TEACH portion can return control to the student portion without knowing the line numbers in advance.
2. Eleven private variables ($, $1,...$0) and a private list ($) are provided for the exclusive use of the TEACH portion. They are initially set to 0 and may be used as counters, etc., knowing that the student portion cannot modify them.
3. Line numbers in the student portion must not exceed 9999. Those in the TEACH portion must be at least 10000.
4. A time limit may be set using TIME. The time is checked at all GØTØ's, IF's and NEXT's; if the time is exceeded, the running stops, thus avoiding a possible infinite loop in the student portion.

As an example, we have included the complete TEACH test program for PIE, the first problem used in the freshman training program. The combined programs are started at line 10000. After setting up certain variables, the test program in line 15000 jumps to the student program. The * indicates the first instruction in the student portion. After the student portion has completed its calculation, the computer "drops down" into the TEACH portion. This time, $9 is > 1 so that we jump to 20000 for checking. We use $9 to count the number of times the TEACH portion is entered--this counter can thus tell us when all testing has been completed, as well as distinguishing between the first and later times the test program is entered.

The block diagram of the test program PIE is also included, and may help in interpreting the program. The five digit numbers in the block diagram refer to line numbers in the test program. Notice that the test program is written entirely in BASIC, except for the special use of $ and *, so that the instructor himself can prepare the test program. When he is ready to allow use of the test program, it is inserted into the TEACH library, from which it cannot be listed.

The following six pages are illustrations of the above-described TEACH program.

The text of this report continues on page 22.

PIE

```
10000 REM  THIS IS A SKELETØN TEACH PRØGRAM.
10010 REM  IT IS DESIGNED BØTH AS A REMINDER, AND TØ SAVE TYPING.
10020 REM  THIS PRØGRAM USES VARIABLES AS FØLLØWS:
10030 REM  $9 (ØNE MØRE THAN NØ. ØF PASSES THRU STUDENT PRØGRAM)
10040 REM  $7 (NØ. ØF TIMES YØU WISH TØ GØ TØ STUDENT PRØGRAM)
10050 REM  $8 (USED IN CHECKING THE AMØUNT HE READS).
10100 REM  READ EACH ØF THE REM'S STARTING IN 11000,
10110 REM  AND INSERT INSTRUCTIØNS IMMEDIATELY FØLLØWING THE REM,
10120 REM  WHENEVER APPRØPRIATE.
11000 REM  DESCRIPTIØN ØF PRØBLEM:
11001 REM  TØ CØMPUTE VALUE ØF PIE.
11002 REM  START WITH HEXAGØN, DØUBLE SIDES  D  TIMES.
11003 REM  DATA: D.  N= NØ. SIDES, S= SIDE, P= APPRØX. PIE.
11500 REM  REMINDER: CHANGE NAME ØF PRØBLEM, USING 'RENAME'.
12000 LET $9 = $9+1
12100 IF $9 > 1 THEN 20000
12200 READ $7
13000 REM  INITIALIZE STUDENT'S VARIABLES TØ RECØGNIZABLE WRØNG NØS.
13010 LET N = -177
13020 LET S = -177
13030 LET P = -277
14000 REM  SET UP YØUR $-VARIABLES. (ELSE THEY WILL BE 0.)
15000 GØTØ *
16000 REM  TIME-LIMIT SET AT 5 SECS. CHANGE IF DESIRED.
16001 TIME 5
20000 IF $9 > 2 THEN 30000
20100 READ $8
20200 IF $8 = 117 THEN 30000
20300 PRINT "YØU ARE READING THE WRØNG AMØUNT ØF DATA."
21000 REM  INSERT AN ADDITIØNAL SENTENCE, IF DESIRED.
21010 PRINT "READ A SINGLE NUMBER D, THEN CØMPUTE N, S, AND P."
22000 STØP
30000 REM  READ A DUPLICATE SET ØF DATA, TØ BE USED IN
30001 REM  PRINTING ERRØR-MESSAGES.
30010 READ $1
31000 REM  READ ØR CØMPUTE CØRRECT ANSWERS.
31010 LET $2 = 6
31020 LET $3 = 1
31030 FØR $ = 1 TØ $1
31040 LET $5 = $3/2
31050 LET $6 = 1-SQR(1-$5↑2)
31060 LET $3 = SQR($5↑2+$6↑2)
31070 LET $2 = 2*$2
31080 LET $4 = $2*$3/2
31090 NEXT $
32000 REM  CHECK STUDENT ANSWERS -- IF ERRØR, THEN 40000.
32010 IF ABS($3-S) > 1E-6 THEN 40000
32020 IF ABS($4-P) > 1E-6 THEN 40000
32030 IF N<>$2 THEN 40000
33000 IF $9 <= $7 THEN *
```

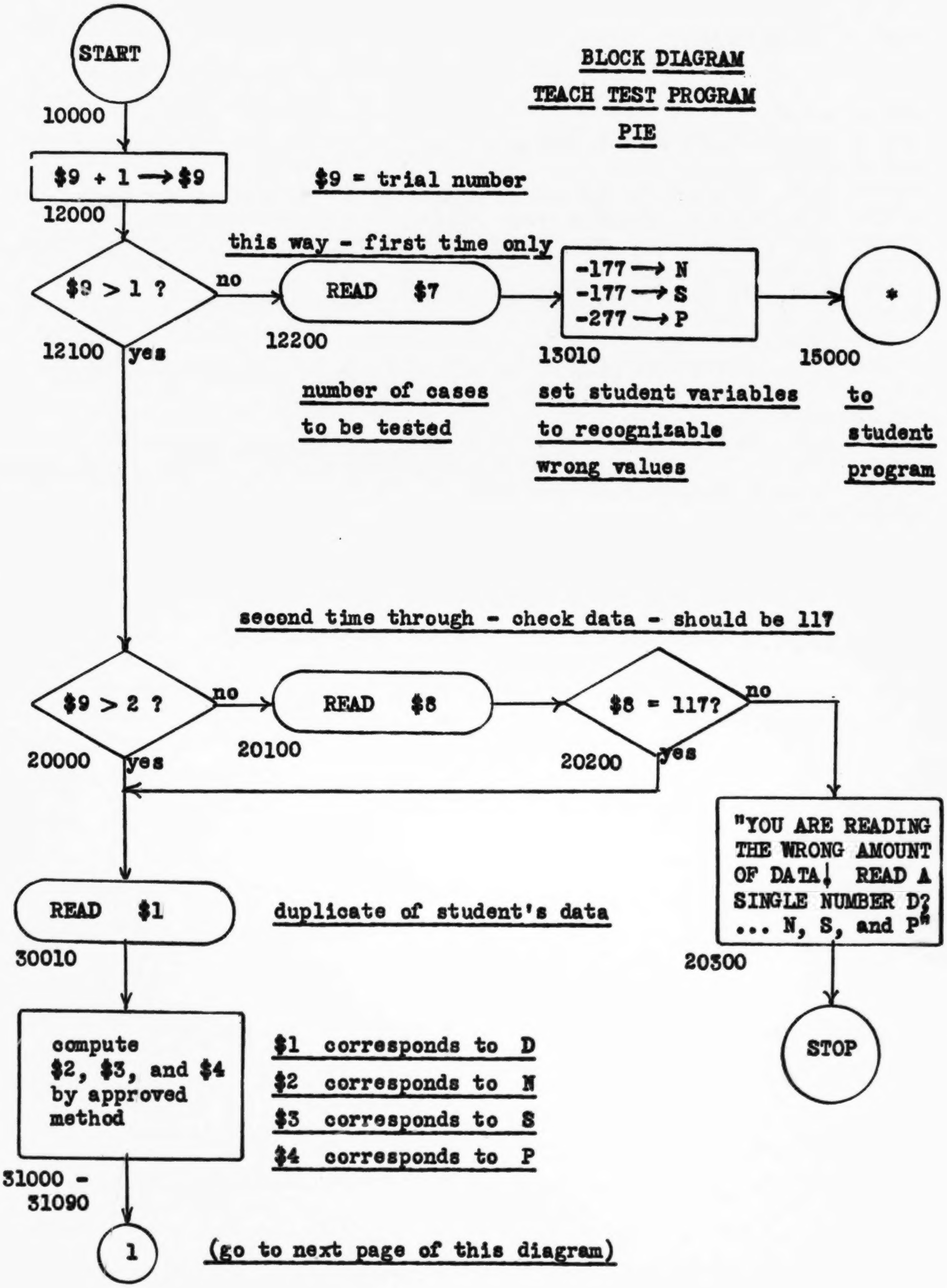
PIE CØNTINUED

```
33100 PRINT "CØNGRATULATIØNS. YØUR PRØGRAM WØRKS."
33200 REM  ADD AN ADDITIØNAL USEFUL ØR FUNNY SENTENCE.
33210 PRINT "YØU HAVE EARNED YØUR FIRST STRIPE AS A PRØGRAMMER."
33220 PRINT
33230 PRINT "HAND IN: THIS PAGE WITH A LIST AND A RUN."
33240 PRINT "BE SURE YØUR NAME AND CØURSE APPEARS ØN IT."
33300 STØP
40000 IF $9 > 2 THEN 50000
41000 REM  CHECK WHETHER ANY ØF THE INITIAL VALUES YØU SUPPLIED
41001 REM  ARE UNCHANGED. IF SØ, STUDENT IS NØT USING THE
41002 REM  CØRRECT VARIABLE(S).
41010 IF N <> -177 THEN 41100
41020 PRINT "YØU ARE NØT USING N FØR THE NUMBER ØF SIDES."
41030 STØP
41100 IF S <> -177 THEN 41150
41110 PRINT "YØU ARE NØT USING S FØR THE LENGTH ØF THE SIDE."
41120 STØP
41150 IF S <> 1 THEN 41200
41160 PRINT "YØU HAVE FAILED TØ ITERATE. "
41170 PRINT "ARE YØU ITERATING D TIMES?"
41180 STØP
41200 IF P <> -277 THEN 42000
41210 PRINT "YØU ARE NØT USING P FØR THE APPRØX. VALUE ØF PIE."
41220 STØP
42000 REM  CHECK WTHETHER WRØNG ANSWER IS RESULT ØF YØUR
42001 REM  INITIAL VALUES. THEN HE IS NØT INITIALIZING.
42100 IF N >= 0 THEN 42200
42110 PRINT "YØU DID NØT INITIALIZE N."
42120 STØP
42200 IF ABS(S-124.4)>1 THEN 50000
42210 PRINT "YØU DID NØT INITIALIZE S."
42220 STØP
50000 REM  DETAILED ERRØR ANALYSIS. THIS IS THE PAY-ØFF.
50010 IF N = $2 THEN 50100
50015 IF $9 > 2 THEN 50150
50020 PRINT "EITHER YØU HAVE INITIALIZED N INCØRRECTLY, ØR ";
50025 IF N > $2 THEN 50050
50030 PRINT "YØU ARE NØT DØUBLING IT."
50040 STØP
50050 PRINT "YØU ARE"
50055 PRINT "ITERATING TØØ MANY TIMES."
50060 STØP
50100 IF S = $3 THEN 50200
50110 IF $9 > 2 THEN 50150
50120 PRINT "YØU ARE CØMPUTING S INCØRRECTLY."
50130 PRINT "PLEASE CHECK THE NØTES HANDED ØUT IN LECTURE."
50140 STØP
50150 PRINT "YØU STARTED ØUT ALL RIGHT, BUT THERE IS AN ERRØR";
50160 PRINT " IN THE WAY YØU ITERATE."
50165 PRINT "ARE YØU ITERATING D TIMES?"
```

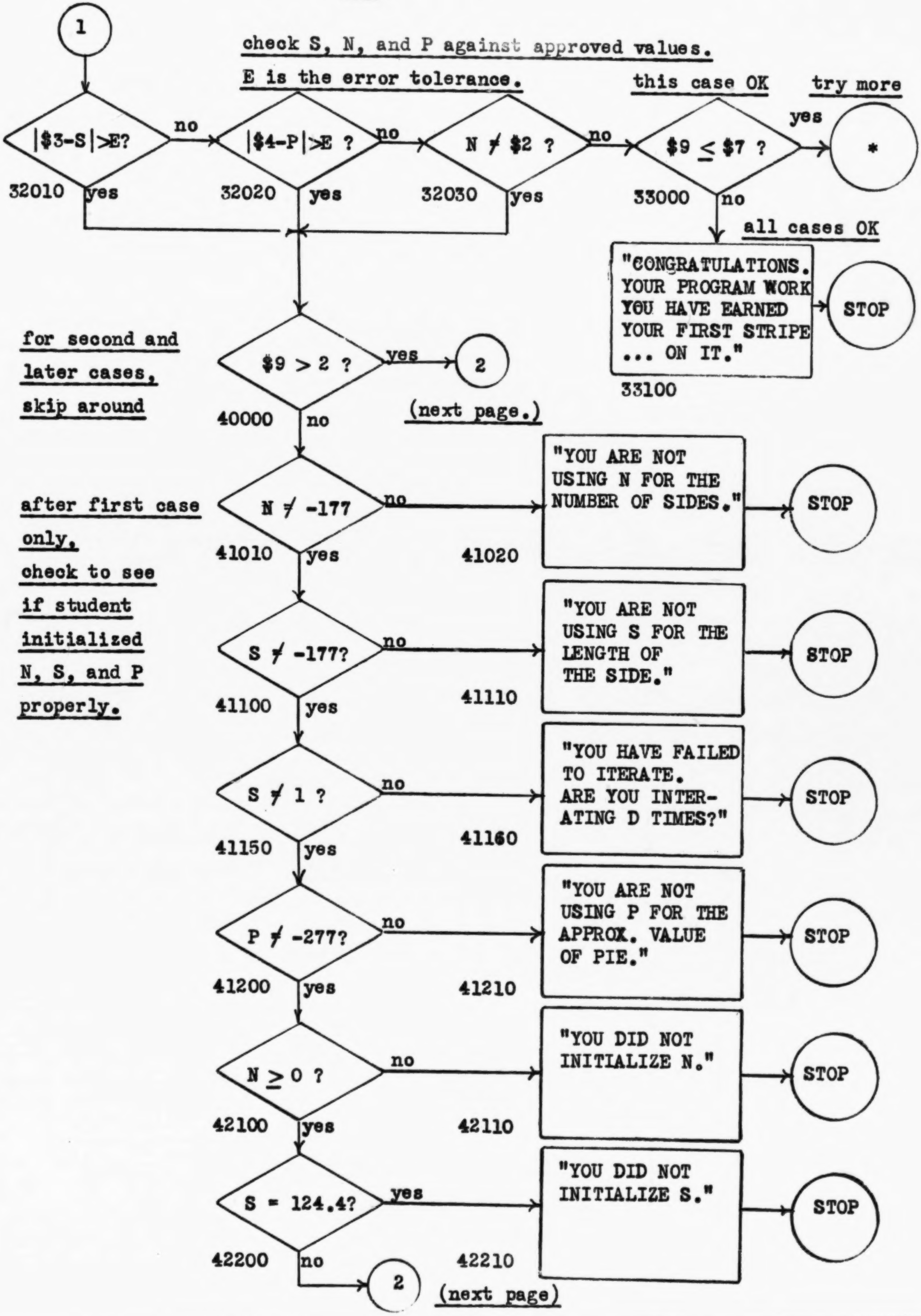
PIE   CØNTINUED

```
50170 STØP
50200 PRINT "YØU ARE FINDING P INCØRRECTLY FRØM N AND S."
50210 STØP
60000 REM  REMINDER: BE SURE THERE IS A 'STØP' AFTER EACH
60001 REM  ERRØR MESSAGE YØU WRØTE.
90000 REM  DATA: NØ. ØF TIMES YØU WANT TØ GØ THRU STUDENT PRØGRAM.
90001 DATA 3
91000 REM  DATA: FIRST SET ØF DATA FØR STUDENT.
91001 DATA 1
92000 DATA 117
93000 REM  DATA: DUPLICATE ØF STUDENT DATA, ØTHER DATA FØR
93001 REM        CHECKING FIRST STUDENT PASS.
93002 DATA 1
94000 REM  DATA: DATA FØR LATER PASSES. ALWAYS HAVE
94001 REM        STUDENT DATA, THEN DUPLICATE, THEN ØTHER DATA.
94002 DATA 2,2,10,10
99999 END
```

BLOCK DIAGRAM
TEACH TEST PROGRAM
PIE


START
10000
$9 + 1 → $9
12000
$9 = trial number
$9 > 1 ?
12100
yes
no
this way - first time only
READ $7
12200
number of cases to be tested
-177 → N
-177 → S
-277 → P
13010
set student variables to recognizable wrong values
*
15000
to student program
second time through - check data - should be 117
$9 > 2 ?
20000
yes
no
READ $8
20100
$8 = 117?
20200
yes
no
"YOU ARE READING THE WRONG AMOUNT OF DATA↓ READ A SINGLE NUMBER D? ... N, S, and P"
20300
STOP
READ $1
30010
duplicate of student's data
compute $2, $3, and $4 by approved method
31000 - 31090
$1 corresponds to D
$2 corresponds to N
$3 corresponds to S
$4 corresponds to P
1
(go to next page of this diagram)

PIE - 2



check S, N, and P against approved values.

E is the error tolerance.

this case OK

try more

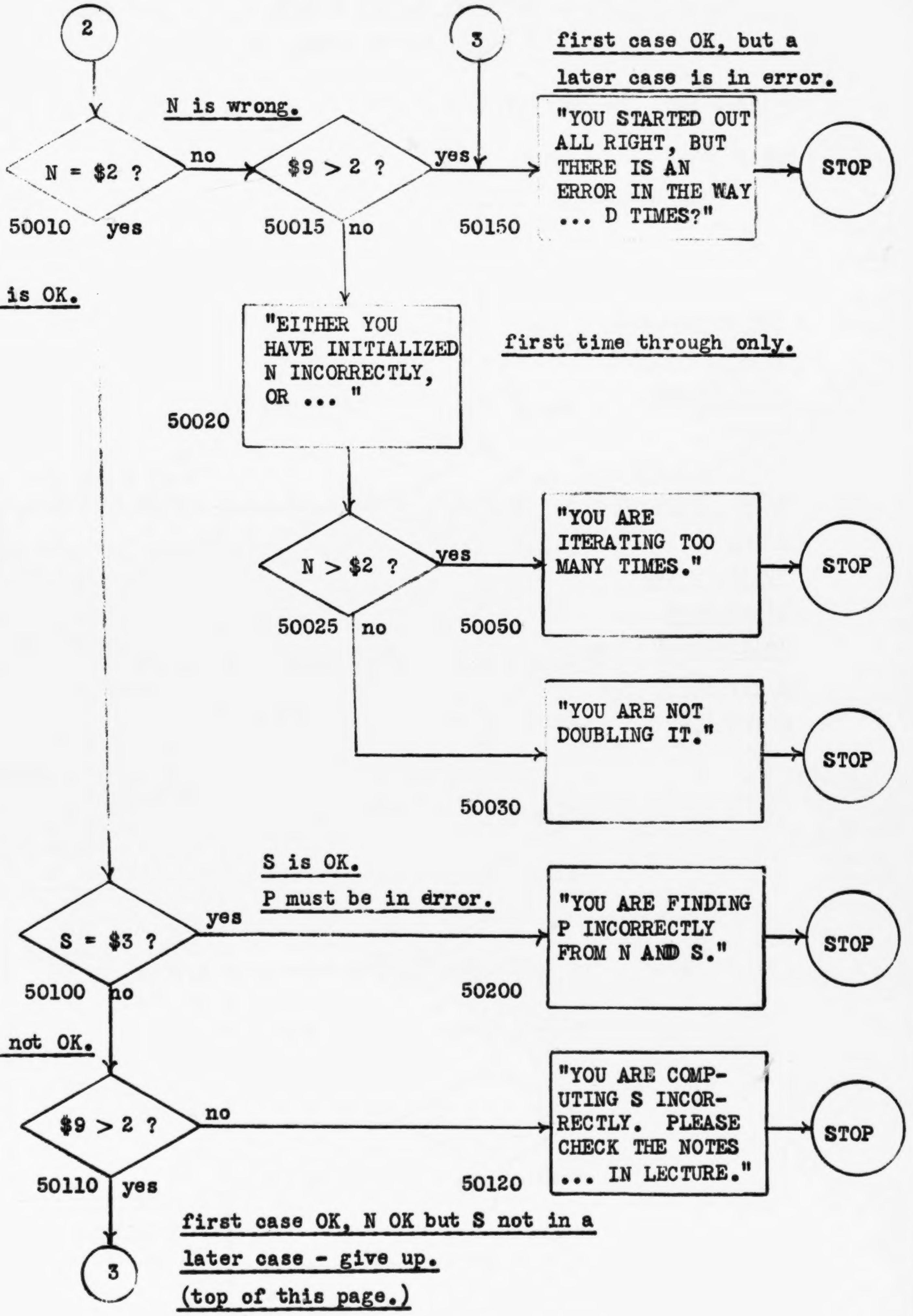
all cases OK

for second and later cases, skip around

after first case only, check to see if student initialized N, S, and P properly.

1
|$3-S|>E?
32010
yes
no
|$4-P|>E ?
32020
yes
no
N ≠ $2 ?
32030
yes
no
$9 ≤ $7 ?
33000
yes
*
no
"CONGRATULATIONS. YOUR PROGRAM WORK YOU HAVE EARNED YOUR FIRST STRIPE ... ON IT."
33100
STOP
$9 > 2 ?
40000
yes
2
(next page.)
no
N ≠ -177
41010
no
"YOU ARE NOT USING N FOR THE NUMBER OF SIDES."
41020
STOP
yes
S ≠ -177?
41100
no
"YOU ARE NOT USING S FOR THE LENGTH OF THE SIDE."
41110
STOP
yes
S ≠ 1 ?
41150
no
"YOU HAVE FAILED TO ITERATE. ARE YOU INTER-ATING D TIMES?"
41160
STOP
yes
P ≠ -277?
41200
no
"YOU ARE NOT USING P FOR THE APPROX. VALUE OF PIE."
41210
STOP
yes
N ≥ 0 ?
42100
no
"YOU DID NOT INITIALIZE N."
42110
STOP
yes
S = 124.4?
42200
yes
"YOU DID NOT INITIALIZE S."
42210
STOP
no
2
(next page)

PIE - 3


2
3
first case OK, but a later case is in error.
N is wrong.
N = $2 ?
no
$9 > 2 ?
yes
"YOU STARTED OUT ALL RIGHT, BUT THERE IS AN ERROR IN THE WAY ... D TIMES?"
STOP
50010
yes
50015
no
50150
N is OK.
"EITHER YOU HAVE INITIALIZED N INCORRECTLY, OR ... "
first time through only.
50020
N > $2 ?
yes
"YOU ARE ITERATING TOO MANY TIMES."
STOP
50025
no
50050
"YOU ARE NOT DOUBLING IT."
STOP
50030
S is OK.
P must be in error.
S = $3 ?
yes
"YOU ARE FINDING P INCORRECTLY FROM N AND S."
STOP
50100
no
50200
S is not OK.
$9 > 2 ?
no
"YOU ARE COMPUTING S INCORRECTLY. PLEASE CHECK THE NOTES ... IN LECTURE."
STOP
50110
yes
50120
3
first case OK, N OK but S not in a later case - give up.
(top of this page.)

TEACH-

```
10000 REM  THIS IS A SKELETØN TEACH PRØGRAM.
10010 REM  IT IS DESIGNED BØTH AS A REMINDER, AND TØ SAVE TYPING.
10020 REM  THIS PRØGRAM USES VARIABLES AS FØLLØWS:
10030 REM  $9 (ØNE MØRE THAN NØ. ØF PASSES THRU STUDENT PRØGRAM)
10040 REM  $7 (NØ. ØF TIMES YØU WISH TØ GØ TØ STUDENT PRØGRAM)
10050 REM  $8 (USED IN CHECKING THE AMØUNT HE READS).
10100 REM  READ EACH ØF THE REM'S STARTING IN 11000,
10110 REM  AND INSERT INSTRUCTIØNS IMMEDIATELY FØLLØWING THE REM,
10120 REM  WHENEVER APPRØPRIATE.
11000 REM  DESCRIPTIØN ØF PRØBLEM:
11500 REM  REMINDER: CHANGE NAME ØF PRØBLEM, USING 'RENAME'.
12000 LET $9 = $9+1
12100 IF $9 > 1 THEN 20000
12200 READ $7
13000 REM  INITIALIZE STUDENT'S VARIABLES TØ RECØGNIZABLE WRØNG NØS.
14000 REM  SET UP YØUR $-VARIABLES. (ELSE THEY WILL BE 0.)
15000 GØTØ *
16000 REM  TIME-LIMIT SET AT 5 SECS. CHANGE IF DESIRED.
16001 TIME 5
20000 IF $9 > 2 THEN 30000
20100 READ $8
20200 IF $8 = 117 THEN 30000
20300 PRINT "YØU ARE READING THE WRØNG AMØUNT ØF DATA."
21000 REM  INSERT AN ADDITIØNAL SENTENCE, IF DESIRED.
22000 STØP
30000 REM  READ A DUPLICATE SET ØF DATA, TØ BE USED IN
30001 REM  PRINTING ERRØR-MESSAGES.
31000 REM  READ ØR CØMPUTE CØRRECT ANSWERS.
32000 REM  CHECK STUDENT ANSWERS -- IF ERRØR, THEN 40000.
33000 IF $9 <= $7 THEN *
33100 PRINT "CØNGRATULATIØNS. YØUR PRØGRAM WØRKS."
33200 REM  ADD AN ADDITIØNAL USEFUL ØR FUNNY SENTENCE.
33300 STØP
40000 IF $9 > 2 THEN 50000
41000 REM  CHECK WHETHER ANY ØF THE INITIAL VALUES YØU SUPPLIED
41001 REM  ARE UNCHANGED. IF SØ, STUDENT IS NØT USING THE
41002 REM  CØRRECT VARIABLE(S).
42000 REM  CHECK WTHETHER WRØNG ANSWER IS RESULT ØF YØUR
42001 REM  INITIAL VALUES. THEN HE IS NØT INITIALIZING.
50000 REM  DETAILED ERRØR ANALYSIS. THIS IS THE PAY-ØFF.
60000 REM  REMINDER: BE SURE THERE IS A 'STØP' AFTER EACH
60001 REM  ERRØR MESSAGE YØU WRØTE.
90000 REM  DATA: NØ. ØF TIMES YØU WANT TØ GØ THRU STUDENT PRØGRAM.
91000 REM  DATA: FIRST SET ØF DATA FØR STUDENT.
92000 DATA 117
93000 REM  DATA: DUPLICATE ØF STUDENT DATA, ØTHER DATA FØR
93001 REM        CHECKING FIRST STUDENT PASS.
94000 REM  DATA: DATA FØR LATER PASSES. ALWAYS HAVE
94001 REM        STUDENT DATA, THEN DUPLICATE, THEN ØTHER DATA.
99999 END
```

To help the instructor get started, we furnish a skeleton TEACH test program containing information remarks and certain common portions of the program that are most likely to be needed. A listing of this skeleton program, called TEACH-,is included. Notice that the time limit is set nominally to 5 seconds in line 16001. It should be emphasized that any statement or variable can be changed-- TEACH- is provided merely to save work.

An illustration of this program is given on the preceding page.

The TEACH system is used for the computer training in freshman mathematics. It is also used by other departments and at other levels for training students. For instance, the Tuck School of Business Administration requires a series of several TEACH-checked exercises for their first year students.

An interesting application is the BASICT system for teaching BASIC in the absence of the introductory lectures. When the user prepares his first (very simple) exercise, he types TEST. If it is accepted, not only is he immediately notified, but also the test program prints out directions for his next exercise. While still in the developmental stage, this approach has already proved quite useful and is continually being improved.

3. A Typical Experience

Following the winter term of 1965-1966, during which the computer training was provided for students enrolled in second-term calculus, a survey of the 166 students in the regular section was conducted. This group excludes the advanced placement and honors students. A completed questionnaire was returned by 96 of the 166 students. The results below are based on this incomplete but nonetheless significant survey.

Based on the survey, the average amount of time spent by the students for the four exercises was:

	Preparation (hours)	teletype
PIE	1.7	1.5
TRAP	1.7	1.5
SINE	1.8	1.5
DIFFEQ*	1.2	1.0

* Only about 63 students had completed DIFFEQ by the time of the survey.

Adding to these average times the two hours of preliminary lectures, we conclude that the total time spent by the "average" student on the entire four-exercise program is around 14 hours. This amount was deemed to be not excessive, and no material was deleted from the mathematics course to compensate.

Other results from the survey show that:

a. A majority felt that two one-hour lectures were either about right or too much.

b. Half the students were not good typists, and half of these felt this caused them to require extra time of up to an hour per problem.

c. There was much discussion between students, but very little use was made of graduate student or faculty consultants.

d. Most, but not all, students felt that the TEACH error messages were helpful, and a few made the excellent observation that they should be suggestive only, allowing the student to figure out his own error.

e. Some improvement in the problem handouts was suggested, but most felt the BASIC Manual was quite good.

f. Some students felt that the computer training helped with their understanding of the Calculus, but many felt that more integration with the course work would be helpful.

For a different term (fall 1965) results were gathered on the numbers of students in several sections who actually completed each of the exercises. In this term, the computer training was required of students in the following courses:

Course Number	Students	Description
4	41	Second term Calculus
5	71	Advanced Placement Calculus
9	50	Honors Section of Math 5
27	18	Sophomore honors (freshman only)

Overall, 95 percent of all exercises were completed successfully. The specific statistics for the four exercises were:

Exercise	Numbers	Percent
PIE	178 out of 181	98
TRAP	172 out of 180	96
SINE	167 out of 180	93
DIFFEQ	168 out of 180	93

While the students were told that the computer exercises were required, no specific penalties for non-completion were mentioned. Even so, the proportion completing this work is considered high in comparison with other types of "required" work, such as daily homework. This indicates to us that the computer training portion of the course is generally popular with the students, and that most of them take it quire seriously.